

```

EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCCCC HHH HHH NNN NNN GGG

```

```
EEEEEEEEEE XX      XX      CCCCCCCC RRRRRRRR TTTTTTTTTT 11      11
EEEEEEEEEE XX      XX      CCCCCCCC RRRRRRRR TTTTTTTTTT 11      11
EE          XX      XX      CC          RR          RR 1111      1111
EE          XX      XX      CC          RR          RR 1111      1111
EE          XX      XX      CC          RR          RR 11      11
EE          XX      XX      CC          RRRRRRRR TT      11      11
EEEEEEEEEE      XX      CC          RRRRRRRR TT      11      11
EEEEEEEEEE      XX      CC          RR      RR TT      11      11
EE          XX      XX      CC          RR      RR TT      11      11
EE          XX      XX      CC          RR      RR TT      11      11
EEEEEEEEEE XX      XX      CCCCCCCC RR      RR TT      111111 111111
EEEEEEEEEE XX      XX      CCCCCCCC RR      RR TT      111111 111111
                                     ....
                                     ....
                                     ....
                                     ....
```

```
LL          IIIIII SSSSSSSS
LL          IIIIII SSSSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SSSSSS
LL          II      SSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
```

```
0001 0 MODULE  exch$rt11                                XTITLE 'RT11 file and directory routines'
0002 0
0003 0      IDENT = 'V04-000'
0004 0      ADDRESSING_MODE (EXTERNAL=LONG_RELATIVE, NONEXTERNAL=WORD_RELATIVE)
0005 0      ) =
0006 1 BEGIN
0007 1
0008 1 *****
0009 1 *
0010 1 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0011 1 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0012 1 *  ALL RIGHTS RESERVED.
0013 1 *
0014 1 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0015 1 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0016 1 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0017 1 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0018 1 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0019 1 *  TRANSFERRED.
0020 1 *
0021 1 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0022 1 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0023 1 *  CORPORATION.
0024 1 *
0025 1 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0026 1 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0027 1 *
0028 1 *
0029 1 *****
0030 1
0031 1 ++
0032 1 FACILITY:      EXCHANGE - Foreign volume interchange facility
0033 1
0034 1 ABSTRACT:      RT-11 volume specific routines
0035 1
0036 1 ENVIRONMENT:    VAX/VMS User mode
0037 1
0038 1 AUTHOR:         CW Hobbs                CREATION DATE: 26-Aug-1982
0039 1
0040 1 MODIFIED BY:
0041 1
0042 1      V03-004  CWH3004      CW Hobbs      25-Jul-1984
0043 1      Move logic check 175 to after a test for global caching,
0044 1      since globally cached write-locked volumes were hitting
0045 1      the trap.
0046 1
0047 1      V03-003  CWH3003      CW Hobbs      12-Apr-1984
0048 1      Disable message about recovering devices, and force
0049 1      /TRANSFER=BLOCK to be global
0050 1
0051 1      V03-002  CWH9001      CW Hobbs      30-Apr-1983
0052 1      Remove debugging call accidentally checked in.
0053 1
0054 1 --
0055 1
0056 1 ! Include files:
0057 1
```


EXCH\$RT11
V04-000

RT11 file and directory routines

E 13

16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32:1

Page 2
(1)

```
: 58      0058 1 MACRO $module_name_string = 'exch$rt11' %;      ! The require file needs to know our module name
: 59      0059 1 REQUIRE 'SRC$:EXCREQ'                          ! Facility-wide require file
: 60      0060 1 ;
```

```
62 0157 1 %SBTTL 'Module table of contents'
63 0158 1
64 0159 1 ! Module table of contents:
65 0160 1
66 0161 1 FORWARD ROUTINE
67 0162 1     exch$rt11_bad_file : NOVALUE,      ! Create a bad file over a bad spot on the volume
68 0163 1     exch$rt11_close_file,          ! RT-11 specific file close routine
69 0164 1     exch$rt11_create_file,         ! RT-11 volume file create
70 0165 1     exch$rt11_delete_file,        ! RT-11 specific file delete routine
71 0166 1     exch$rt11_dircache_exit_handler : NOVALUE, ! Routine to flush the cache
72 0167 1     exch$rt11_dircache_start : NOVALUE,      ! Engage write-back caching on the directory
73 0168 1     exch$rt11_dircache_stop : NOVALUE,       ! Disengage caching and flush the directory
74 0169 1     exch$rt11_dirseg_flush,       ! Write out directory segments
75 0170 1     exch$rt11_dirseg_get,         ! Return pointer to specific directory segment
76 0171 1     exch$rt11_dirseg_get_nochk : jsb_r1r2,    ! Return pointer to directory segment without checking
77 0172 1     exch$rt11_dirseg_put,        ! Write a specific directory segment
78 0173 1     exch$rt11_expand_filename,    ! Convert directory entry to ASCII filename
79 0174 1     exch$rt11_format_current_date : NOVALUE jsb_r1, ! Put the current date into a directory entry
80 0175 1     exch$rt11_mount,             ! RT-11 specific volume mount routine
81 0176 1     exch$rt11_open_file,         ! RT-11 specific file open routine
82 0177 1     exch$rt11_write_cleanup : NOVALUE,      ! Finish writing to the volume
83 0178 1     exch$rt11_write_prepare : NOVALUE,      ! Prepare to write to the volume
84 0179 1     exch$rt11_zero_marks : NOVALUE,        ! Clear marks on volume
85 0180 1
86 0181 1
87 0182 1 ! EXCHANGE facility routines
88 0183 1
89 0184 1 EXTERNAL ROUTINE
90 0185 1     exch$cmd_fetch_recfmt_implied : NOVALUE, ! Get or assume the value for /RECORD_FORMAT
91 0186 1     exch$cmd_match_filename,        ! Compare expanded file names for match
92 0187 1     exch$cmd_related_file_parse,   ! Build an output file name
93 0188 1     exch$io_rt11_read,             ! Read blocks from a random access device
94 0189 1     exch$io_rt11_write,           ! Write blocks to a random access device
95 0190 1     exch$pdg_filter_filename,     ! Remove invalid characters from a filename
96 0191 1     exch$pdg_flush_write_buffer,  ! Flush any records waiting in the write buffer
97 0192 1     exch$pdg_get,                ! Get functions for small PDP record structure
98 0193 1     exch$pdg_put,                ! Put functions for small PDP record structure
99 0194 1     exch$rtacp_check_position : NOVALUE, ! Find directory entry if it has moved
100 0195 1     exch$rtacp_clean_directory,   ! Shuffle and/or split directories as needed
101 0196 1     exch$rtacp_find_empty_area,   ! Compress directory structure and find free space
102 0197 1     exch$rtacp_find_file,        ! RT-11 directory search routine
103 0198 1     exch$rtacp_verify_directory, ! Verify directory structure and compute volume size
104 0199 1     exch$util_fao_buffer,        ! Do an FAO conversion
105 0200 1     exch$util_file_error,        ! Signal an RMS error
106 0201 1     exch$util_radix50_from_ascii, ! Convert characters to Radix-50 from Ascii
107 0202 1     exch$util_radix50_to_ascii,   ! Convert characters from Radix-50 to Ascii
108 0203 1     exch$util_rt11ctx_allocate,   ! Get an RT-11 context block
109 0204 1     exch$util_rt11ctx_release : NOVALUE, ! Give it back
110 0205 1     exch$util_vm_allocate,        ! Get some virtual memory
111 0206 1     exch$util_vm_allocate_zeroed, ! Get some virtual memory, initialized to zero
112 0207 1     exch$util_vm_release,        ! Return some virtual memory
113 0208 1
114 0209 1
115 0210 1 ! Equated symbols:
116 0211 1
117 0212 1 ! LITERAL
118 0213 1
```

EXCHSRT11
V04-000

RT11 file and directory routines
Module table of contents

G 13
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 4
(2)

:	119	0214	1	
:	120	0215	1	: Bound declarations:
:	121	0216	1	:
:	122	0217	1	: BIND
:	123	0218	1	: ;


```
125 0219 1 GLOBAL ROUTINE exch$rt11_bad_file (filb : $ref_bblock) : NOVALUE = %SBTTL 'exch$rt11_bad_file (filb)'
126 0220 2 BEGIN
127 0221 2 ++
128 0222 2
129 0223 2 FUNCTIONAL DESCRIPTION:
130 0224 2
131 0225 2 Perform RT-11 bad block handling by placing a FILE.BAD file over the bad block. This routine will b
132 0226 2 called when a bad block is detected on the output file during a copy operation. We assume that ther
133 0227 2 a zero-block empty file entry following the current entry.
134 0228 2
135 0229 2 One of the following actions will be taken:
136 0230 2
137 0231 2 The bad block is the first block in the output file:
138 0232 2 The output file is renamed to a 1 block FILE.BAD and made permanent. Remaining bloc
139 0233 2 moved to the empty file. (Single block files are treated as this case)
140 0234 2
141 0235 2 The bad block is in the middle of the output file:
142 0236 2 The output file will be left as a tentative file. If there is room for another entr
143 0237 2 the current directory segment, then a 1 block FILE.BAD is created and the remaining
144 0238 2 blocks are moved to a newly created empty file. If there is no room to add an entry
145 0239 2 FILE.BAD will contain all the free blocks in addition to the one known to be bad.
146 0240 2
147 0241 2 The bad block is at the end of the file:
148 0242 2 The output file will be left as a tentative file, and a 1 block FILE.BAD will be cre
149 0243 2
150 0244 2 INPUT/OUTPUT:
151 0245 2
152 0246 2 filb - pointer to block describing the file
153 0247 2
154 0248 2 IMPLICIT INPUTS:
155 0249 2
156 0250 2 things hanging from the filb, notably the bad block number is in RAB$L_BKT in the volb rab.
157 0251 2
158 0252 2 OUTPUTS:
159 0253 2
160 0254 2 filb - receive info pertaining to the file to be closed
161 0255 2
162 0256 2 IMPLICIT OUTPUTS:
163 0257 2
164 0258 2 none
165 0259 2
166 0260 2 ROUTINE VALUE:
167 0261 2
168 0262 2 none
169 0263 2
170 0264 2 SIDE EFFECTS:
171 0265 2
172 0266 2 none
173 0267 2 --
174 0268 2
175 0269 2 $dbgtrc_prefix ('exch$rt11_bad_file> ');
176 0270 2
177 0271 2 LOCAL
178 0272 2 bad_pbn,
179 0273 2 blks_before,
180 0274 2 blks_after,
181 0275 2 ent_len,
```

```
! pbn of the bad block
! blocks before the bad block
! blocks after the bad one
! length of a directory entry
```

```
182 0276 2 emp : $ref_bblock, ! pointer to the empty entry after this one
183 0277 eos : $ref_bblock, ! pointer to the end of segment marker
184 0278 status
185 0279 ;
186 0280
187 0281 BIND
188 0282 copy = exch$a_gbl [excg$a_copy_work]: $ref_bblock,
189 0283 ctx = filb [filb$a_context] : $ref_bblock,
190 0284 namb = filb [filb$a_assoc_namb] : $ref_bblock,
191 0285 volb = filb [filb$a_assoc_volb] : $ref_bblock,
192 0286 rab = volb [volb$a_rab] : $ref_bblock,
193 0287 seg = ctx [rt11ctx$a_seg_address] : $ref_bblock, ! pointer to the directory segment
194 0288 ent = ctx [rt11ctx$a_ent_address] : $ref_bblock ! and the directory entry for this file
195 0289 ;
196 0290
197 0291 $debug_print_lit ('entry');
198 0292
199 0293 $block_check (2, .filb, filb, 565); !?? definitely over-zealous checking
200 0294 $block_check (2, .namb, namb, 566);
201 0295 $block_check (2, .volb, volb, 567);
202 0296 $block_check (2, .ctx, rt11ctx, 568);
203 0297 $logic_check (2, (.ctx [rt11ctx$a_assoc_filb] EQL .filb), 217);
204 0298 $logic_check (2, (.ctx [rt11ctx$a_assoc_volb] EQL .volb), 218);
205 0299 $logic_check (2, (.ctx [rt11ctx$a_output_file]), 219);
206 0300 $logic_check (3, (exch$rtacp_verify_directory (.volb)), 220);
207 0301
208 0302 ! The bad block number is sitting in the rab
209 0303
210 0304 bad_pbn = .rab [rab$a_bkt];
211 0305 IF .volb [volb$a_virtual] ! Undo pbn -> vbn mapping
212 0306 THEN
213 0307 bad_pbn = .bad_pbn - 1;
214 0308 $logic_check (2, ((.bad_pbn GEQU .ctx [rt11ctx$a_start_block]) AND (.bad_pbn LEQU .ctx [rt11ctx$a_eof_block])
215 0309 $trace_print_fao ('bad_pbn !UL', .bad_pbn);
216 0310
217 0311 ! Let the outer routines know that we have erased this file
218 0312
219 0313 filb [filb$a_file_erased] = true;
220 0314
221 0315 ! Get the pointer to the empty entry after this one
222 0316
223 0317 ent_len = rt11ent$a_length + .seg [rt11hdr$a_extra_bytes];
224 0318 emp = .ent + .ent_len;
225 0319 $logic_check (3, ((.emp [rt11ent$a_type_byte] EQL rt11ent$a_type_empty) AND (.emp [rt11ent$a_blocks] EQL 0)),
226 0320 $logic_check (3, ((.ent [rt11ent$a_type] EQL rt11ent$a_type_tentative) AND (.ent [rt11ent$a_blocks] NEQ 0)),
227 0321
228 0322 ! A structured GOTO follows. EXITLOOPS will be used to rejoin common code at the end of the routine
229 0323
230 0324 WHILE 1
231 0325 DO
232 0326 BEGIN
233 0327
234 0328 ! How many blocks were written before the bad block
235 0329
236 0330 blks_before = .bad_pbn - .ctx [rt11ctx$a_start_block];
237 0331
238 0332 ! If blocks before is zero, we can tie this off right now
```



```
239 0333 3 !
240 0334 3 !
241 0335 3 !
242 0336 4 !
243 0337 4 !
244 0338 4 !
245 0339 4 !
246 0340 4 !
247 0341 4 !
248 0342 4 !
249 0343 4 !
250 0344 4 !
251 0345 4 !
252 0346 4 !
253 0347 4 !
254 0348 4 !
255 0349 4 !
256 0350 4 !
257 0351 4 !
258 0352 4 !
259 0353 4 !
260 0354 4 !
261 0355 4 !
262 0356 4 !
263 0357 4 !
264 0358 4 !
265 0359 4 !
266 0360 4 !
267 0361 4 !
268 0362 4 !
269 0363 4 !
270 0364 4 !
271 0365 4 !
272 0366 4 !
273 0367 4 !
274 0368 4 !
275 0369 4 !
276 0370 4 !
277 0371 4 !
278 0372 4 !
279 0373 4 !
280 0374 4 !
281 0375 4 !
282 0376 4 !
283 0377 4 !
284 0378 4 !
285 0379 4 !
286 0380 4 !
287 0381 4 !
288 0382 4 !
289 0383 4 !
290 0384 4 !
291 0385 4 !
292 0386 4 !
293 0387 4 !
294 0388 4 !
295 0389 4 !

!
IF .blks_before EQL 0
THEN
  BEGIN
    ! Move any remaining blocks to the empty entry
    emp [rt11ent$w_blocks] = .ent [rt11ent$w_blocks] - 1;
    ! Create a one block permanent FILE.BAD in the entry
    ent [rt11ent$w_blocks] = 1;
    $exch_signal (exch$rt11_badfile, 1, .bad_pbn);          ! Tell the guy that we made a .BAD file
    EXITLOOP;          ! Done here, jump to the end to fill in the rest of the bad
  END;

! How many blocks are after the bad one
blks_after = .ctx [rt11ctx$L_eof_block] - .bad_pbn;
! If blocks after is zero, we can also do the work and exit
IF .blks_after EQL 0
THEN
  BEGIN
    ! Remove one block from the tentative file
    ent [rt11ent$w_blocks] = .ent [rt11ent$w_blocks] - 1;
    ! Move the empty pointer to the ent pointer, where the common code expects to find the bad entry
    ent = .emp;
    ! Create a one block permanent FILE.BAD in the empty entry
    ent [rt11ent$w_blocks] = 1;
    $exch_signal (exch$rt11_badfile, 1, .bad_pbn);          ! Tell the guy
    EXITLOOP;
  END;

! OK, the bad block is in the middle of the tentative file. We have two choices now, depending on wheth
! have room in the segment to add another file entry. First, find the end of segment marker.
eos = .emp;          ! Point to the empty entry
WHILE 1
DO
  BEGIN
    eos = .eos + .ent len;          ! Advance to the next entry
    $logic_check (2, 7.eos LSSU (.seg + rt11$k_dirseglen), 224);
    IF .eos [rt11ent$v_type] EQL rt11ent$m_typ_end_segment
    THEN
      EXITLOOP
```

```
296 0390 3      END;
297 0391 3
298 0392 3      ! Make sure that there is room to add one more entry to this segment.  If not, we will have to add a big
299 0393 3
300 0394 4      IF ((.eos+2 + .ent_len) GEQU (.seg + rt11$dirseglen))
301 0395 3      THEN
302 0396 4      BEGIN
303 0397 4
304 0398 4          ! Make the tentative file include all the blocks before the bad one
305 0399 4          ent [rt11ent$w_blocks] = .blks_before;
306 0400 4
307 0401 4          ! Move the empty pointer to the ent pointer, where the common code expects to find the bad entry
308 0402 4
309 0403 4          ent = .emp;
310 0404 4
311 0405 4          ! Put the rest in a permanent FILE.BAD in the empty entry
312 0406 4
313 0407 4          ent [rt11ent$w_blocks] = .blks_after + 1;
314 0408 4
315 0409 4          $exch_signal (exch$rt11_badfile, 1, .bad_pbn, exch$rt11_bigbadfile); ! Tell the guy the bad news
316 0410 4
317 0411 4      EXITLOOP;
318 0412 4      END
319 0413 4
320 0414 4      ! Room for another entry, make it <TENT> <BAD> <EMPTY>
321 0415 4
322 0416 4      ELSE
323 0417 3      BEGIN
324 0418 4      LOCAL
325 0419 4      sl;
326 0420 4
327 0421 4          ! Slide the rest of the segment up one entry to make room for the new entry
328 0422 4
329 0423 4          sl = .eos+2 - .emp;
330 0424 4          CH$MOVE (.sl, .emp, .emp + .ent_len); ! Length of segment between empty and end
331 0425 4          ! Slide rest of segment up
332 0426 4
333 0427 4          ! Finish up the tentative entry
334 0428 4          ent [rt11ent$w_blocks] = .blks_before;
335 0429 4
336 0430 4          ! Point "ent" at the bad entry and "emp" at the new empty
337 0431 4
338 0432 4          ent = .emp;
339 0433 4          emp = .emp + .ent_len;
340 0434 4
341 0435 4          ! Finish up the new empty entry.  Since we slid the old empty up, all we need to do is set the lengt
342 0436 4          emp [rt11ent$w_blocks] = .blks_after;
343 0437 4          $logic_check (3, (.emp [rt11ent$b_type_byte] EQL rt11ent$m_typ_empty), 225);
344 0438 4
345 0439 4          ! Create a one block permanent FILE.BAD in the middle entry.
346 0440 4
347 0441 4          ent [rt11ent$w_blocks] = 1;
348 0442 4
349 0443 4          $exch_signal (exch$rt11_badfile, 1, .bad_pbn); ! Tell the guy
350 0444 4
351 0445 4
352 0446 4
```

```
353 0447 4 EXITLOOP;
354 0448 4 END;
355 0449 4 $logic_check (0, (false), 216); ! We should have hit an exitloop before here
356 0450 4 END;
357 0451 4
358 0452 4 ! "ent" points to the bad entry, fill in the info common to all three cases
359 0453 4
360 0454 4 ent [rt1lent$b_type_byte] = rt1lent$m_typ_permanent;
361 0455 4 ent [rt1lent$l_filename] = r50_file; ! 'FILE' in radix 50
362 0456 4 ent [rt1lent$w_filetype] = r50_bad; ! 'BAD' in radix 50
363 0457 4 exch$rt11_format_current_date (.ent);
364 0458 4
365 0459 4 exch$rt11_dirseg_put (.volb, .ctx [rt1lctx$l_seg_number]); ! Flush the modified segment
366 0460 4
367 0461 4 ! Force a directory update on disk if caching is active
368 0462 4
369 0463 4 IF .volb [volb$v_dircache_active]
370 0464 4 THEN
371 0465 4 BEGIN
372 0466 4 exch$rt11_dircache_stop (.volb); ! Do the I/O
373 0467 4 exch$rt11_dircache_start (.volb); ! Renable caching
374 0468 4 END;
375 0469 4
376 0470 4 ! Set a flag so that the copy command will attempt to retry the current file
377 0471 4
378 0472 4 copy [copy$v_reopen_input] = true;
379 0473 4
380 0474 4 RETURN;
381 0475 4
382 0476 4 1 END;
```

```
.TITLE EXCH$RT11 RT11 file and directory routines
.IDENT \V04-000\
```

```
.EXTRN EXCH$CMD_FETCH_RECMT IMPLIED
.EXTRN EXCH$CMD_MATCH_FILENAME
.EXTRN EXCH$CMD_RELATED_FILE_PARSE
.EXTRN EXCH$IO_RT11_READ
.EXTRN EXCH$IO_RT11_WRITE
.EXTRN EXCH$PDP_FILTER_FILENAME
.EXTRN EXCH$PDP_FLUSH_WRITE_BUFFER
.EXTRN EXCH$PDP_GET, EXCH$PDP_PUT
.EXTRN EXCH$RTACP_CHECK_POSITION
.EXTRN EXCH$RTACP_CLEAN_DIRECTORY
.EXTRN EXCH$RTACP_FIND_EMPTY_AREA
.EXTRN EXCH$RTACP_FIND_FILE
.EXTRN EXCH$RTACP_VERIFY_DIRECTORY
.EXTRN EXCH$UTIL_FAO_BUFFER
.EXTRN EXCH$UTIL_FILE_ERROR
.EXTRN EXCH$UTIL_RADIX50_FROM_ASCII
.EXTRN EXCH$UTIL_RADIX50_TO_ASCII
.EXTRN EXCH$UTIL_RT11CTX_ALLOCATE
.EXTRN EXCH$UTIL_RT11CTX_RELEASE
.EXTRN EXCH$UTIL_VM_ALLOCATE
.EXTRN EXCH$UTIL_VM_ALLOCATE_ZEROED
.EXTRN EXCH$UTIL_VM_RELEASE
```


PC	OP	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	OP10	OP11	OP12	OP13	OP14	OP15	OP16	OP17	OP18	OP19	OP20	OP21	OP22	OP23	OP24	OP25	OP26	OP27	OP28	OP29	OP30	OP31	OP32	OP33	OP34	OP35	OP36	OP37	OP38	OP39	OP40	OP41	OP42	OP43	OP44	OP45	OP46	OP47	OP48	OP49	OP50	OP51	OP52	OP53	OP54	OP55	OP56	OP57	OP58	OP59	OP60	OP61	OP62	OP63	OP64	OP65	OP66	OP67	OP68	OP69	OP70	OP71	OP72	OP73	OP74	OP75	OP76	OP77	OP78	OP79	OP80	OP81	OP82	OP83	OP84	OP85	OP86	OP87	OP88	OP89	OP90	OP91	OP92	OP93	OP94	OP95	OP96	OP97	OP98	OP99	OP100	OP101	OP102	OP103	OP104	OP105	OP106	OP107	OP108	OP109	OP110	OP111	OP112	OP113	OP114	OP115	OP116	OP117	OP118	OP119	OP120	OP121	OP122	OP123	OP124	OP125	OP126	OP127	OP128	OP129	OP130	OP131	OP132	OP133	OP134	OP135	OP136	OP137	OP138	OP139	OP140	OP141	OP142	OP143	OP144	OP145	OP146	OP147	OP148	OP149	OP150	OP151	OP152	OP153	OP154	OP155	OP156	OP157	OP158	OP159	OP160	OP161	OP162	OP163	OP164	OP165	OP166	OP167	OP168	OP169	OP170	OP171	OP172	OP173	OP174	OP175	OP176	OP177	OP178	OP179	OP180	OP181	OP182	OP183	OP184	OP185	OP186	OP187	OP188	OP189	OP190	OP191	OP192	OP193	OP194	OP195	OP196	OP197	OP198	OP199	OP200	OP201	OP202	OP203	OP204	OP205	OP206	OP207	OP208	OP209	OP210	OP211	OP212	OP213	OP214	OP215	OP216	OP217	OP218	OP219	OP220	OP221	OP222	OP223	OP224	OP225	OP226	OP227	OP228	OP229	OP230	OP231	OP232	OP233	OP234	OP235	OP236	OP237	OP238	OP239	OP240	OP241	OP242	OP243	OP244	OP245	OP246	OP247	OP248	OP249	OP250	OP251	OP252	OP253	OP254	OP255	OP256	OP257	OP258	OP259	OP260	OP261	OP262	OP263	OP264	OP265	OP266	OP267	OP268	OP269	OP270	OP271	OP272	OP273	OP274	OP275	OP276	OP277	OP278	OP279	OP280	OP281	OP282	OP283	OP284	OP285	OP286	OP287	OP288	OP289	OP290	OP291	OP292	OP293	OP294	OP295	OP296	OP297	OP298	OP299	OP300	OP301	OP302	OP303	OP304	OP305	OP306	OP307	OP308	OP309	OP310	OP311	OP312	OP313	OP314	OP315	OP316	OP317	OP318	OP319	OP320	OP321	OP322	OP323	OP324	OP325	OP326	OP327	OP328	OP329	OP330	OP331	OP332	OP333	OP334	OP335	OP336	OP337	OP338	OP339	OP340	OP341	OP342	OP343	OP344	OP345	OP346	OP347	OP348	OP349	OP350	OP351	OP352	OP353	OP354	OP355	OP356	OP357	OP358	OP359	OP360	OP361	OP362	OP363	OP364	OP365	OP366	OP367	OP368	OP369	OP370	OP371	OP372	OP373	OP374	OP375	OP376	OP377	OP378	OP379	OP380	OP381	OP382	OP383	OP384	OP385	OP386	OP387	OP388	OP389	OP390	OP391	OP392	OP393	OP394	OP395	OP396	OP397	OP398	OP399	OP400	OP401	OP402	OP403	OP404	OP405	OP406	OP407	OP408	OP409	OP410	OP411	OP412	OP413	OP414	OP415	OP416	OP417	OP418	OP419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

			00000000G	8F	DD	0000D	PUSHL	#EXCH\$ BADLOGIC	
			00000000G	03	FB	000E3	CALLS	#3, LIB\$STOP	
			28	A3	88	000EA	BISB2	#4, 43(R3)	0313
			08	AE	D0	000EE	MOVL	122(R8), 8(SP)	0317
	50		08	AE	C1	000F3	ADDL3	#6, 8(SP), R0	
			04	AE	3C	000F8	MOVZWL	(R0), ENT_LEN	
			04	AE	C0	000FC	ADDL2	#14, ENT_LEN	
	57		6A	04	C1	00100	ADDL3	ENT_LEN, (R10), EMP	0318
OC	AE		5B	72	C3	00105	SUBL3	114(R8), BAD_PBN, BLKS_BEFORE	0330
				52	D4	0010B	CLRL	R2	0334
				OC	AE	D5	TSTL	BLKS_BEFORE	
				0E	12	00110	BNEQ	7\$	
				52	D6	00112	INCL	R2	
				6A	D0	00114	MOVL	(R10), R0	0340
08	A7	08	A0	01	A3	00117	SUBW3	#1, 8(R0), 8(EMP)	
				00AD	31	0011D	BRW	13\$	0344
10	AE	20	A8	5B	C3	00120	SUBL3	BAD_PBN, 32(R8), BLKS_AFTER	0353
			OC	52	E9	00126	BLBC	R2, -8\$	0357
			50	6A	D0	00129	MOVL	(R10), R0	0363
				08	A0	B7	DECW	8(R0)	
			6A	57	D0	0012F	MOVL	EMP, (R10)	0367
				0095	31	00132	BRW	12\$	0371
				56	D0	00135	MOVL	EMP, EOS	0381
			04	AE	C0	00138	ADDL2	ENT_LEN, EOS	0385
50		08	AE	8F	C1	0013C	ADDL3	#1024, 8(SP), R0	0386
			50	56	D1	00145	CMPL	EOS, R0	
				13	1F	00148	BLSSU	10\$	
			7E	E0	8F	9A	MOVZBL	#224, -(SP)	
				01	DD	0014E	PUSHL	#1	
				8F	DD	00150	PUSHL	#EXCH\$ BADLOGIC	
			00000000G	03	FB	00156	CALLS	#3, LIB\$STOP	
08	01	A6	04	00	ED	0015D	CMPZV	#0, #4, 1(EOS), #8	0387
				D3	12	00163	BNEQ	9\$	
				02	C1	00165	ADDL3	#2, ENT_LEN, R0	0394
			04	AE	50	C1	ADDL3	R0, EOS, R1	
			56	50	C1	0016A	ADDL3	#1024, 8(SP), R0	
			08	AE	8F	C1	ADDL3	R1, R0	
			50	50	D1	00177	CMPL	11\$	
				2D	1F	0017A	BLSSU		
			50	6A	D0	0017C	MOVL	(R10), R0	0400
			08	A0	AE	B0	MOVW	BLKS_BEFORE, 8(R0)	
			6A	57	D0	00184	MOVL	EMP, (R10)	0404
			50	6A	D0	00187	MOVL	(R10), R0	0408
08	A0	10	AE	01	A1	0018A	ADDW3	#1, BLKS_AFTER, 8(R0)	
				8F	DD	00190	PUSHL	#EXCH\$ RT11_BIGBADFILE	0410
				5B	DD	00196	PUSHL	BAD_PBN	
				01	DD	00198	PUSHL	#1	
				8F	DD	0019A	PUSHL	#EXCH\$ RT11_BADFILE	
			00000000G	04	FB	001A0	CALLS	#4, LIB\$SIGNAL	
				39	11	001A7	BRB	14\$	0396
				57	C3	001A9	SUBL3	EMP, EOS, R0	0424
			50	02	C0	001AD	ADDL2	#2, SL	
			67	50	28	001B0	MOVW	SL, (EMP), @ENT_LEN[EMP]	0425
04	BE47		50	6A	D0	001B6	MOVL	(R10), R0	0429
			08	A0	AE	B0	MOVW	BLKS_BEFORE, 8(R0)	
			6A	57	D0	001BE	MOVL	EMP, (R10)	0433
			57	04	AE	C0	ADDL2	ENT_LEN, EMP	0434
			08	A7	10	AE	MOVW	BLKS_AFTER, 8(EMP)	0438

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_bad_file (filb)

B 14
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 12
(3)

08	50		6A	DD	001CA	12\$:	MOVL	(R10), R0	:	0443
	A0		01	DD	001CD	13\$:	MOVW	#1, 8(R0)	:	
			5B	DD	001D1		PUSHL	BAD_PBN	:	0445
			01	DD	001D3		PUSHL	#1	:	
00000000G	00	00000000G	8F	DD	001D5		PUSHL	#EXCH\$ RT11 BADFILE	:	
	51		03	FB	001DB		CALLS	#3, LIB\$SIGNAL	:	
01	A1		6A	DD	001E2	14\$:	MOVL	(R10), R1	:	0454
02	A1	1F4026F4	04	90	001E5		MOVB	#4, 1(R1)	:	
06	A1	OCAC	8F	DD	001E9		MOVL	#524297972, 2(R1)	:	0455
			8F	DD	001F1		MOVW	#3244, 6(R1)	:	0456
			0000V	30	001F7		BSBW	EXCH\$RT11_FORMAT_CURRENT_DATE	:	0457
		76	A8	DD	001FA		PUSHL	118(R8)	:	0459
			59	DD	001FD		PUSHL	R9	:	
0000V	CF		02	FB	001FF		CALLS	#2, EXCH\$RT11_DIRSEG_PUT	:	
	OE	50	A9	E9	00204		BLBC	80(R9), 15\$:	0463
			59	DD	00208		PUSHL	R9	:	0466
0000V	CF		01	FB	0020A		CALLS	#1, EXCH\$RT11_DIRCACHE_STOP	:	
			59	DD	0020F		PUSHL	R9	:	0467
0000V	CF		01	FB	00211		CALLS	#1, EXCH\$RT11_DIRCACHE_START	:	
	50	00	BE	DD	00216	15\$:	MOVL	@0(SP), R0	:	0472
34	A0		04	88	0021A		BISB2	#4, 52(R0)	:	
			04	00	0021E		RET		:	0476

; Routine Size: 543 bytes, Routine Base: EXCH\$RT11_CODE + 0000


```
384 0477 1 GLOBAL ROUTINE exch$rt11_close_file (filb : $ref_bblock) = %SBTTL 'exch$rt11_close_file (filb)'
385 0478 2 BEGIN
386 0479 3 ++
387 0480 4
388 0481 5 FUNCTIONAL DESCRIPTION:
389 0482 6
390 0483 7     Perform RT-11 volume specific close processing
391 0484 8
392 0485 9 INPUT/OUTPUT:
393 0486 10
394 0487 11     filb - pointer to block describing the file
395 0488 12
396 0489 13 IMPLICIT INPUTS:
397 0490 14
398 0491 15     none
399 0492 16
400 0493 17 OUTPUTS:
401 0494 18
402 0495 19     filb - receive info pertaining to the file to be closed
403 0496 20
404 0497 21 IMPLICIT OUTPUTS:
405 0498 22
406 0499 23     none
407 0500 24
408 0501 25 ROUTINE VALUE:
409 0502 26
410 0503 27     true if able to close the file, false otherwise
411 0504 28
412 0505 29 SIDE EFFECTS:
413 0506 30
414 0507 31     none
415 0508 32
416 0509 33 --
417 0510 34 $dbgtrc_prefix ('exch$rt11_close_file> ');
418 0511 35
419 0512 36 LOCAL
420 0513 37     status
421 0514 38 ;
422 0515 39
423 0516 40 BIND
424 0517 41     ctx = filb [filb$a_context] : $ref_bblock,
425 0518 42     namb = filb [filb$a_assoc_namb] : $ref_bblock,
426 0519 43     volb = filb [filb$a_assoc_volb] : $ref_bblock
427 0520 44 ;
428 0521 45
429 0522 46 $debug_print_lit ('entry');
430 0523 47
431 0524 48 $block_check (2, .filb, filb, 447);
432 0525 49 $block_check (2, .namb, namb, 448);
433 0526 50 $block_check (2, .volb, volb, 449);
434 0527 51 $block_check (2, .ctx, rt11ctx, 451);
435 0528 52 $logic_check (2, (.ctx [rt11ctx$a_assoc_filb] EQL .filb), 122);
436 0529 53 $logic_check (2, (.ctx [rt11ctx$a_assoc_volb] EQL .volb), 123);
437 0530 54 $logic_check (5, (exch$rtacp_verify_directory (.volb)), 186);
```

!?? definitely over-zealous checking

```
439 0531 2 ! Output files need some directory tweaks and a buffer flush
440 0532 2
441 0533 2 IF .ctx [rt11ctx$v_output_file]
442 0534 2 THEN
443 0535 2 BEGIN
444 0536 2 LOCAL
445 0537 2     blks_used;
446 0538 2     emp : $ref_bblock;          ! pointer to the empty entry after this one
447 0539 2 BIND
448 0540 2     seg = ctx [rt11ctx$a_seg_address] : $ref_bblock,      ! pointer to a directory segment
449 0541 2     ent = ctx [rt11ctx$a_ent_address] : $ref_bblock;      ! and the directory entry for this file
450 0542 2
451 0543 2 ! Flush any blocks that are sitting in the output buffer
452 0544 2
453 0545 2 IF NOT (status = exch$dpd_flush_write_buffer (.ctx))
454 0546 2 THEN
455 0547 2     RETURN .status;
456 0548 2
457 0549 2 ! How many blocks were actually used in the file
458 0550 2
459 0551 2 blks_used = 1 + .ctx [rt11ctx$l_high_block_written] - .ctx [rt11ctx$l_start_block];
460 0552 2
461 0553 2 ! If an allocation was requested, and the allocation was more than was used, then increase the size to t
462 0554 2 allocation request. The extra blocks will be filled with nulls.
463 0555 2
464 0556 2 IF ((.filb [filb$l_q_allocation] NEQ 0)
465 0557 2 AND
466 0558 2     (.filb [filb$l_q_allocation] GTRU .blks_used))
467 0559 2 THEN
468 0560 2 BEGIN
469 0561 2 LOCAL
470 0562 2     blk_cnt,
471 0563 2     blks_to_clear,
472 0564 2     cur_blk;
473 0565 2
474 0566 2 ! Figure out how many blocks to clear and the pbn of the first block to clear
475 0567 2
476 0568 2 blks_to_clear = .filb [filb$l_q_allocation] - .blks_used;          ! Number of null blocks to w
477 0569 2 cur_blk = .ctx [rt11ctx$l_high_block_written] + 1;                ! Block at which to write nu
478 0570 2 $logic check (3, (.ctx [rt11ctx$a_buffer] NEQ 0), 195);
479 0571 2 CH$FILE (0, ctx$k_buffer_length, .ctx [rt11ctx$a_buffer]);        ! Fill the buffer with nulls
480 0572 2
481 0573 2 ! Write the null blocks
482 0574 2
483 0575 2 blk_cnt = .blks_to_clear;
484 0576 2 WHILE .blk_cnt GTR 0          ! Note the signed compare
485 0577 2 DO
486 0578 2 BEGIN
487 0579 2 IF NOT (status = exch$io_rt11_write (.volb,
488 0580 2     .cur_blk,
489 0581 2     MINU (.blk_cnt, ctx$k_buffer_blocks),
490 0582 2     .ctx [rt11ctx$a_buffer]))
491 0583 2 THEN
492 0584 2 BEGIN
493 0585 2     exch$rt11_bad_file (.filb);
494 0586 2 RETURN .status;
495 0587 2 END;
```

```
496 0588 5
497 0589
498 0590
499 0591
500 0592
501 0593
502 0594
503 0595
504 0596
505 0597
506 0598
507 0599
508 0600
509 0601
510 0602
511 0603
512 0604
513 0605
514 0606
515 0607
516 0608
517 0609
518 P 0610
519 0611
520 0612
521 0613
522 0614
523 0615
524 0616
525 0617
526 0618
527 0619
528 0620
529 0621
530 0622
531 0623
532 0624
533 0625
534 0626
535 0627
536 0628
537 P 0629
538 0630
539 0631
540 0632
541 0633
542 0634
543 0635
544 0636
545 0637
546 0638
547 0639
548 0640
549 0641
550 0642
551 0643
552 0644

! Point at the next chunk to write
blk_cnt = .blk_cnt - ctx$buffer_blocks;
cur_blk = .cur_blk + ctx$buffer_blocks;
END;

! Update the pointers to the new highest block written
ctx [rt11ctx$l_high_block_written] = .ctx [rt11ctx$l_high_block_written] + .blks_to_clear;
blks_used = .blks_used + .blks_to_clear;

END;

! Truncate the file by moving any unused blocks to the empty directory entry which immediately follows t
! entry.
emp = .ent + rt11ent$length + .seg [rt11hdr$w_extra_bytes];
! Pointer to the empty entry
$logic_check (3, ((.emp [rt11ent$b_type_byte] EQL rt11ent$m_typ_empty) AND (.emp [rt11ent$w_blocks] EQL
emp [rt11ent$w_blocks] = .ctx [rt11ctx$l_eof_block] -
! Count of leftovers (0 is o
.ctx [rt11ctx$l_high_block_written]);
ent [rt11ent$w_blocks] = .blks_used;
$debug_print_fao ('used !UL, Left !UL, eof !UL, high !UL', .ent [rt11ent$w_blocks], .emp [rt11ent$w_b
.ctx [rt11ctx$l_eof_block], .ctx [rt11ctx$l_high_block_written]);

! If there is another file with the same name around, we need to delete it now
IF .filb [filb$w_delete_previous]
THEN
BEGIN
LOCAL
ctx2 : $ref_bblock;

ctx2 = exch$util_rt11ctx_allocate (.volb, 0); ! Get an RT11 context block

IF exch$rtacp_find_file (.ctx2, ctx [rt11ctx$w_exp_fullname], .ctx [rt11ctx$l_exp_fullname_len])
THEN
BEGIN
BIND
copy = exch$a_gbl [excg$a_copy_work] : $ref_bblock,
ent2 = ctx2 [rt11ctx$a_ent_address] : $ref_bblock;
$debug_print_fao ('deleting previous copy of "TAF"',
.filb [filb$l_result_name_len], filb [filb$w_result_name]);
$logic_check (2, (NOT .ctx2 [rt11ctx$w_typ_protected]), T72); ! Must be able to delete
ent2 [rt11ent$b_type_byte] = rt11ent$m_typ_empty; ! It is gone
exch$rt11_dirseg_put (.volb, .ctx2 [rt11ctx$l_seg_number]);
IF .copy [copy$w_q_log]
THEN
sexch_signal (exch$deleteprev, 2, .filb [filb$l_result_name_len], filb [filb$w_result_name])
END
ELSE
$logic_check (3, (false), 171);

! Return the context block
exch$util_rt11ctx_release (.ctx2);
```



```
.. 553 0645      END;
.. 554 0646
.. 555 0647      ! And finally, mark our new file as permanent and current, and write the directory segment
.. 556 0648      !
.. 557 0649      ent [rt1lent$y_type] = rt1lent$m_typ_permanent;      ! Mark only type field, protected bit might be set
.. 558 0650      ent [rt1lent$b_job] = 1;      ! Mark the entry as current
.. 559 0651      exch$rt11_dirseg_put (.volb, .ctx [rt1lctx$L_seg_number]);
.. 560 0652
.. 561 0653      $logic_check (4, (exch$rtacp_verify_directory (.volb)), 187);
.. 562 0654      END;
.. 563 0655
.. 564 0656      ! Clear the stream active bit and all other context flags
.. 565 0657      !
.. 566 0658      ctx [rt1lctx$L_flags] = 0;
.. 567 0659
.. 568 0660      RETURN true;
.. 569 0661      END;
```

```
                                .EXTRN  EXCH$_DELETEPREV
                                .ENTRY   EXCH$RT11_CLOSE_FILE, Save R2,R3,R4,R5,R6,- : 0477
                                R7,R8,R9,R10,R11
                                SUBL2    #4, SP
                                MOVL     FILB, R7
                                MOVL     #56295674, R2
                                MOVZWL   #447, R1
                                MOVL     R7, R0
                                JSB      EXCH$UTIL_BLOCK_CHECK
                                MOVL     #17432823, R2
                                MOVZWL   #448, R1
                                MOVL     24(R7), R0
                                JSB      EXCH$UTIL_BLOCK_CHECK
                                MOVL     28(R7), (SP)
                                MOVL     #68878579, R2
                                MOVZWL   #449, R1
                                MOVL     (SP), R0
                                JSB      EXCH$UTIL_BLOCK_CHECK
                                MOVL     32(R7), R6
                                MOVL     #8519924, R2
                                MOVZWL   #451, R1
                                MOVL     R6, R0
                                JSB      EXCH$UTIL_BLOCK_CHECK
                                CMPL     16(R6), R7
                                BEQL     1$
                                MOVZBL   #122, -(SP)
                                PUSHL    #1
                                PUSHL    #EXCH$_BADLOGIC
                                CALLS    #3, LIB$STOP
                                CMPL     20(R6), (SP)
                                BEQL     2$
                                MOVZBL   #123, -(SP)
                                PUSHL    #1
                                PUSHL    #EXCH$_BADLOGIC
                                CALLS    #3, LIB$STOP
                                BBS      #1, 40(R6), 3$
                                : 0517
                                : 0524
                                : 0525
                                : 0526
                                : 0527
                                : 0528
                                : 0529
                                : 0533
```

		OFFC 00000		
5E		04	C2	00002
57	04	AC	DO	00005
52	035B00FA	8F	DO	00009
51	01BF	8F	3C	00010
50		57	DO	00015
	00000000G	EF	16	00018
52	010A00F7	8F	DO	0001E
51	01C0	8F	3C	00025
50	18	A7	DO	0002A
	00000000G	EF	16	0002E
6E	1C	A7	DO	00034
52	041B00F3	8F	DO	00038
51	01C1	8F	3C	0003F
50		6E	DO	00044
	00000000G	EF	16	00047
56	20	A7	DO	0004D
52	008200F4	8F	DO	00051
51	01C3	8F	3C	00058
50		56	DO	0005D
	00000000G	EF	16	00060
57	10	A6	D1	00066
		13	13	0006A
7E	7A	8F	9A	0006C
		01	DD	00070
	00000000G	8F	DD	00072
00		03	FB	00078
6E	14	A6	D1	0007F 1\$:
		13	13	00083
7E	7B	8F	9A	00085
		01	DD	00089
	00000000G	8F	DD	0008B
03	00000000G	03	FB	00091
28	A6	01	EO	00098 2\$:

				013E	31	0009D	BRW	15\$	
				56	DD	000A0	PUSHL	R6	0545
		00000000G	EF	01	FB	000A2	CALLS	#1, EXCH\$PDP_FLUSH_WRITE_BUFFER	
			5A	50	DD	000A9	MOVL	R0, STATUS	
			54	5A	E9	000AC	BLBC	STATUS, 6\$	
	50	34	A6	72	A6	C3	SUBL3	114(R6), 52(R6), R0	0551
			58	01	A0	9E	MOVAB	1(R0), BLKS_USED	
			50	2D	A7	DD	MOVL	45(R7), R0	0556
					57	13	BEQL	9\$	
			58	50	D1	000BF	CMPL	R0, BLKS_USED	0558
					52	1B	BLEQU	9\$	
	59		50	5B	C3	000C4	SUBL3	BLKS_USED, R0, BLKS_TO_CLEAR	0568
	58	34	A6	01	C1	000C8	ADDL3	#1, 52(R6), CUR_BLK	0569
1800	8F	00	6E	00	2C	000CD	MOVCS	#0, (SP), #0, #8144, @24(R6)	0571
				18	B6	000D4			
			52	59	DD	000D6	MOVL	BLKS_TO_CLEAR, BLK_CNT	0575
				52	D5	000D9	TSTL	BLK_CNT	0576
				32	15	000DB	BLEQ	8\$	
				18	A6	DD	PUSHL	24(R6)	0582
					52	DD	PUSHL	BLK_CNT	0581
			0C	6E	D1	000E2	CMPL	(SP), #12	
			6E	03	1B	000E5	BLEQU	5\$	
				0C	DD	000E7	MOVL	#12, (SP)	
				58	DD	000EA	PUSHL	CUR_BLK	0580
				0C	AE	DD	PUSHL	12(SP)	0579
		00000000G	EF	04	FB	000EF	CALLS	#4, EXCH\$IO_RT11_WRITE	
			5A	50	DD	000F6	MOVL	R0, STATUS	
			0B	5A	E8	000F9	BLBS	STATUS, 7\$	
				57	DD	000FC	PUSHL	R7	0585
		FCDE	CF	01	FB	000FE	CALLS	#1, EXCH\$RT11_BAD_FILE	
			50	5A	DD	00103	MOVL	STATUS, R0	0586
					04	00106	RET		
			52	0C	C2	00107	SUBL2	#12, BLK_CNT	0591
			58	0C	C0	0010A	ADDL2	#12, CUR_BLK	0592
				CA	11	0010D	BRB	4\$	0576
		34	A6	59	C0	0010F	ADDL2	BLKS_TO_CLEAR, 52(R6)	0597
			58	59	C0	00113	ADDL2	BLKS_TO_CLEAR, BLKS_USED	0598
			53	7E	A6	DD	MOVL	126(R6), R3	0605
			50	7A	A6	DD	MOVL	122(R6), R0	
			50	06	A0	3C	MOVZWL	6(R0), R0	
			50	0E	A340	9E	MOVAB	14(R3)(R0), EMP	
08	A0	20	A6	34	A6	A3	SUBW3	52(R6), 32(R6), 8(EMP)	0608
		08	A3		5B	B0	MOVW	BLKS_USED, 8(R3)	0609
	03	2B	A7		06	E0	BBS	#6, 43(R7), 10\$	0615
					008F	31	BRW	14\$	
					7E	D4	CLRL	-(SP)	0621
				04	AE	DD	PUSHL	4(SP)	
		00000000G	EF		02	FB	CALLS	#2, EXCH\$UTIL_RT11CTX_ALLOCATE	
			52		50	DD	MOVL	R0, CTX2	
				46	A6	DD	PUSHL	70(R6)	0623
				54	A6	9F	PUSHAB	84(R6)	
					52	DD	PUSHL	CTX2	
		00000000G	EF		03	FB	CALLS	#3, EXCH\$RTACP_FIND_FILE	
			52		50	E9	BLBC	R0, 12\$	
54	00000000G	EF			04	C1	ADDL3	#4, EXCH\$A_GBL, R4	0627
				39	A2	95	TSTB	57(CTX2)	0631
					13	1B	BGEQ	11\$	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_close_file (filb)

H 14
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 18
(5)

		7E	AC	8F	9A	00168	MOVZBL	#172, -(SP)	
				01	DD	0016C	PUSHL	#1	
			00000000G	8F	DD	0016E	PUSHL	#EXCH\$ BADLOGIC	
	00000000G	00		03	FB	00174	CALLS	#3, LIB\$STOP	
		50	7E	A2	DD	0017B	11\$:	MOVL	126(CTX2), R0
	01	A0		02	90	0017F	MOVB	#2, 1(R0)	0632
			76	A2	DD	00183	PUSHL	118(CTX2)	0633
			04	AE	DD	00186	PUSHL	4(SP)	
	0000V	CF		02	FB	00189	CALLS	#2, EXCH\$RT11_DIRSEG_PUT	
		50		64	DD	0018E	MOVL	(R4), R0	0634
2A	30	A0		03	E1	00191	BBC	#3, 48(R0), 13\$	
			5A	A7	9F	00196	PUSHAB	90(R7)	0636
			3A	A7	DD	00199	PUSHL	58(R7)	
				02	DD	0019C	PUSHL	#2	
	00000000G	00	00000000G	8F	DD	0019E	PUSHL	#EXCH\$ DELETEPREV	
				04	FB	001A4	CALLS	#4, LIB\$SIGNAL	
		7E	AB	13	11	001AB	BRB	13\$	0623
				8F	9A	001AD	12\$:	MOVZBL	#171, -(SP)
				01	DD	001B1	PUSHL	#1	0639
			00000000G	8F	DD	001B3	PUSHL	#EXCH\$ BADLOGIC	
	00000000G	00		03	FB	001B9	CALLS	#3, LIB\$STOP	
				52	DD	001C0	13\$:	PUSHL	CTX2
	00000000G	EF		01	FB	001C2	CALLS	#1, EXCH\$UTIL_RT11CTX_RELEASE	0643
01	A3	00		04	F0	001C9	14\$:	INSV	#4, #0, #4, 1(R3)
		0B	A3	01	90	001CF	MOVB	#1, 11(R3)	0649
			76	A6	DD	001D3	PUSHL	118(R6)	0650
			04	AE	DD	001D6	PUSHL	4(SP)	0651
	0000V	CF		02	FB	001D9	CALLS	#2, EXCH\$RT11_DIRSEG_PUT	
			28	A6	D4	001DE	15\$:	CLRL	40(R6)
		50		01	DD	001E1	MOVL	#1, R0	0658
				04	001E4	RET			0660
									0661

: Routine Size: 485 bytes, Routine Base: EXCH\$RT11_CODE + 021F


```
571 0662 1 GLOBAL ROUTINE exch$rt11_create_file = %SBTTL 'exch$rt11_create_file'
572 0663 2 BEGIN
573 0664 3 ++
574 0665 4
575 0666 5 FUNCTIONAL DESCRIPTION:
576 0667 6
577 0668 7     Perform RT-11 volume specific create processing
578 0669 8
579 0670 9 INPUT:
580 0671 10
581 0672 11     none
582 0673 12
583 0674 13 IMPLICIT INPUTS:
584 0675 14
585 0676 15     copy [copy$a_out_filb] - pointer to the filb for the output file
586 0677 16     copy [copy$a_inp_filb] - pointer to the filb for the input file
587 0678 17
588 0679 18 OUTPUTS:
589 0680 19
590 0681 20     none
591 0682 21
592 0683 22 IMPLICIT OUTPUTS:
593 0684 23
594 0685 24     copy [copy$a_out_filb] - block receives info pertaining to the created file
595 0686 25
596 0687 26
597 0688 27 ROUTINE VALUE:
598 0689 28
599 0690 29     true if able to create a file, false otherwise
600 0691 30
601 0692 31 SIDE EFFECTS:
602 0693 32
603 0694 33     none
604 0695 34
605 0696 35 --
606 0697 36 $dbgtrc_prefix ('rt11_create_file> ');
607 0698 37
608 0699 38 LOCAL
609 0700 39     rfp : $bblock [nam$a_bln+nam$a_maxrss],
610 0701 40     nam_len,
611 0702 41     typ_len,
612 0703 42     tot_len,
613 0704 43     ent : $ref_bblock,
614 0705 44     start_block,
615 0706 45     blocks,
616 0707 46     physical,
617 0708 47     status
618 0709 48     ;
619 0710 49
620 0711 50 BIND
621 0712 51     copy      = exch$a_gbl [excg$a_copy_work]      : $ref_bblock,
622 0713 52     out_name  = copy [copy$a_output_filename]      : $desc_block,
623 0714 53     inp_filb  = copy [copy$a_inp_filb]              : $ref_bblock,
624 0715 54     inp_namb  = inp_filb [filb$a_assoc_namb]        : $ref_bblock,
625 0716 55     inp_ctx   = inp_filb [filb$a_context]           : $ref_bblock,
626 0717 56     out_filb  = copy [copy$a_out_filb]              : $ref_bblock,
627 0718 57     out_namb  = out_filb [filb$a_assoc_namb]        : $ref_bblock,
```

```
.. 628      0719 2      out_ctx = out_filb [filb$a_context]      : $ref_bblock.  
.. 629      0720 2      volb   = out_filb [filb$a_assoc_volb]   : $ref_bblock.  
.. 630      0721 2      ;  
.. 631      0722 2      ;  
.. 632      0723 2      $debug_print_lit ('entry');  
.. 633      0724 2      ;  
.. 634      0725 2      $block_check (1, .out_filb, filb, 426);  
.. 635      0726 2      $block_check (1, .inp_filb, filb, 536);  
.. 636      0727 2      $block_check (2, .out_namb, namb, 491);  
.. 637      0728 2      $block_check (2, .inp_namb, namb, 492);  
.. 638      0729 2      $block_check (2, .volb, volb, 531);  
.. 639      0730 2      $logic_check (4, (exch$rtacp_verify_directory (.volb)), 188);
```

```
641 0731 2 ! Make certain that write access is permitted, this should be checked long before we get here
642 0732
643 0733 $logic_check (1, (.volb [volb$v_write]), 166);
644 0734
645 0735 ! If the context pointer is null, then allocate and initialize it.
646 0736
647 0737 IF .out_ctx EQL 0
648 0738 THEN
649 0739     out_ctx = exch$util_rt11ctx_allocate (.volb, .out_filb) ! Get an RT11 context block
650 0740
651 0741 ELSE
652 0742     $block_check (2, .out_ctx, rt11ctx, 534); ! Make sure that it is what we think it is
653 0743
654 0744 ! Make sure that we haven't crossed signals someplace
655 0745
656 0746 $logic_check (4, (.out_ctx [rt11ctx$a_assoc_filb] EQL .out_filb), 162);
657 0747 $logic_check (4, (.out_ctx [rt11ctx$a_assoc_volb] EQL .volb), 163);
658 0748
659 0749 ! Set the rest of the block to nulls, nothing carries over from one output file to the next
660 0750
661 0751 CH$FILL (0, rt11ctx$k_end_zero - rt11ctx$k_start_zero, ! Set rest of block to nulls
662 0752     .out_ctx + rt11ctx$k_start_zero);
663 0753
664 0754 ! Perform an RMS output file parse on the related name (the result name for the input file) and the
665 0755 ! requested output name from the command line.
666 0756
667 0757 IF NOT (status = exch$cmd_related_file_parse (
668 0758     .out_name [dsc$a_length], .out_name [dsc$a_pointer], ! Command line out p
669 0759     .inp_filb [filb$_result_name_len], inp_filb [filb$_result_name], ! Related name
670 0760     rfp)) ! Gets new name
671 0761 THEN
672 0762     BEGIN
673 0763
674 0764     ! Move the raw name to where it is accessible for the outer signal
675 0765
676 0766     CH$MOVE (.out_name [dsc$a_length], .out_name [dsc$a_pointer], out_filb [filb$_result_name]);
677 0767     RETURN .status;
678 0768
679 0769     END;
680 0770
681 0771 ! Create the result file name in the filb
682 0772
683 0773 out_filb [filb$v_name_change] = false; ! Assume no name change
684 0774 tot_len = .rfp [nam$b_name]; ! Remember starting length
685 0775 rfp [nam$b_name] = exch$dpdp_filter_filename (.rfp [nam$b_name], .rfp [nam$l_name]); ! Remove invalid cha
686 0776 nam_len = MINU (.rfp [nam$b_name], 6); ! Maximum name is six letters
687 0777 IF .tot_len NEQ .nam_len ! If final length not same as initial, then it has changed
688 0778 THEN
689 0779     out_filb [filb$v_name_change] = true;
690 0780
691 0781 tot_len = .rfp [nam$b_type]; ! Remember starting length of type field
692 0782 rfp [nam$b_type] = 1 + exch$dpdp_filter_filename (.rfp [nam$b_type] - 1, .rfp [nam$l_type] + 1);
693 0783 typ_len = MINU (.rfp [nam$b_type], 4); ! Maximum type is three (plus the separating dot)
694 0784 IF .tot_len NEQ .typ_len
695 0785 THEN
696 0786     out_filb [filb$v_name_change] = true;
697 0787
```



```
698 0788 2 tot_len = .nam_len + .typ_len; ! Final length of both
699 0789 out_filb [filb$L_result_name_len] = .volb [volb$L_vol_ident_len] + .tot_len; ! Length of volume ident
700 0790 $logic_check (2, (.out_filb [filb$L_result_name_len] [EQU filb$$result_name], 164));
701 0791 CHSCOPY (.volb [volb$L_vol_ident_len], volb [volb$L_vol_ident], ! Volume name
702 0792 .nam_len, .rfp [nam$L_name], .typ_len, .rfp [nam$L_type],
703 0793 0, filb$$result_name, out_filb [filb$L_result_name]);
704 0794
705 0795 ! Do not create a .BAD file unless this is an explicit copy
706 0796
707 0797 IF CH$EQL (4, UPLIT BYTE ('.BAD'), .typ_len, .rfp [nam$L_type])
708 0798 THEN
709 0799 BEGIN
710 0800 IF .inp_namb [namb$V_wild_name] OR .inp_namb [namb$V_wild_type] ! Wild in input?
711 0801 OR
712 0802 .out_namb [namb$V_wild_name] OR .out_namb [namb$V_wild_type] ! Wild in output?
713 0803 THEN
714 0804 RETURN exch$_nocopbad;
715 0805 END;
716 0806
717 0807 $debug_print_fao ('Looking for "!AF"', .out_filb [filb$L_result_name_len], out_filb [filb$L_result_name]);
718 0808
719 0809 ! See if we will have to delete a same-named file later on
720 0810
721 0811 out_filb [filb$V_delete_previous] = false; ! Assume we won't have to delete
722 0812 IF exch$rtacp_find_file (.out_ctx, out_filb [filb$L_result_name] + .volb [volb$L_vol_ident_len], .tot_len)
723 0813 THEN
724 0814 BEGIN
725 0815 LOCAL
726 0816 retstat, ! Help BLISS figure out the two signals are the same
727 0817 sigstat;
728 0818
729 0819 $debug_print_fao ('file "!AF" exists', .out_filb [filb$L_result_name_len], out_filb [filb$L_result_name])
730 0820 $logic_check (2, (.inp_ctx NEQ 0), 209);
731 0821
732 0822 ! If the input and output files are identical during a wildcard copy, this is illegal
733 0823
734 0824 IF .inp_ctx [rt11ctx$b_type] EQL exchblk$k_rt11ctx ! Input must also be RT-11
735 0825 THEN
736 0826
737 0827 ! If the directory entry addresses are the same, this is in fact the same file. Note that we are
738 0828 ! assuming that nothing has happened which might have restructured the directory.
739 0829
740 0830 IF .inp_ctx [rt11ctx$a_ent_address] EQL .out_ctx [rt11ctx$a_ent_address]
741 0831 THEN
742 0832 IF .inp_namb [namb$V_wild_name] OR .inp_namb [namb$V_wild_type] ! Wild in input?
743 0833 OR
744 0834 .copy [copy$V_q_replace]
745 0835 THEN
746 0836 RETURN exch$_nocopsamdev;
747 0837
748 0838 ! Verify that it is ok to delete the existing file
749 0839
750 0840 IF .out_ctx [rt11ctx$b_job] NEQ 0 ! Can't delete protected files
751 0841 THEN
752 0842 RETURN exch$_nocopdup;
753 0843
754 0844 IF .out_ctx [rt11ctx$V_typ_protected] ! Can't delete protected files
```

```
755      0845 3 THEN
756      0846 RETURN exch$_nocopprot;
757      0847
758      0848 IF NOT .copy [copy$_v_q_delete] ! /NODELETE has been requested, don't do it
759      0849 THEN
760      0850 RETURN exch$_nocopnodel;
761      0851
762      0852 IF .out_ctx [rt11ctx$_w_filetype] EQL r50_bad ! Cannot delete a file with .BAD extension during a
763      0853 THEN
764      0854 RETURN exch$_nocopbaddel;
765      0855
766      0856 IF .out_ctx [rt11ctx$_w_filetype] EQL r50_sys ! Cannot delete a file with .SYS extension during a
767      0857 AND ! unless
768      0858 NOT .copy [copy$_v_q_system] ! /SYSTEM has been specified
769      0859 THEN
770      0860 RETURN exch$_nocopsysdel;
771      0861
772      0862 ! If a delete-before-write operation is requested, delete this file now
773      0863
774      0864 IF .copy [copy$_v_q_replace]
775      0865 THEN
776      0866 BEGIN
777      0867 BIND
778      0868 ent = out_ctx [rt11ctx$_a_ent_address] : $ref_bblock;
779      0869 P $debug_print_fao ('deleting previous copy of "'AF'";
780      0870 4 out_filb [filb$_l_result_name_len], out_filb [filb$_t_result_name
781      0871 4 $logic_check (2, (NOT .out_ctx [rt11ctx$_v_typ_protected]), 179); ! Must be able to delete
782      0872 4 ent [rt11ent$_b_type_byte] = rt11ent$_m_ttyp_empty; ! It is gone
783      0873 4 exch$rt11_dirseg_put (.volb, .out_ctx [rt11ctx$_l_seg_number]); ! Write the directory segment
784      0874 4 IF .copy [copy$_v_q_log]
785      0875 4 THEN
786      0876 4 $exch_signal (exch$_deleteprev, 2, .out_filb [filb$_l_result_name_len], out_filb [filb$_t_result_n
787      0877 4 END
788      0878
789      0879 ! otherwise remember that we have some extra work to do when we close the file
790      0880
791      0881 ELSE
792      0882 out_filb [filb$_v_delete_previous] = true;
793      0883
794      0884 END;
795      0885
796      0886 ! Reset the rest of the block to nulls, nothing carries over from before
797      0887
798      0888 CH$FILL (0, rt11ctx$_k_end_zero - rt11ctx$_k_start_zero, ! Set rest of block to nulls
799      0889 .out_ctx + rt11ctx$_k_start_zero);
800      0890
801      0891 ! If a /ALLOCATION qualifier has been seen, use that value. If BLOCKS ends up with the value 0, then
802      0892 ! we will get the largest area on the volume.
803      0893
804      0894 blocks = (IF .inp_filb [filb$_l_q_allocation] NEQ 0 ! If specified on the input
805      0895 THEN ! then
806      0896 .inp_filb [filb$_l_q_allocation] ! use that quantity
807      0897 ELSE ! otherwise
808      0898 .copy [copy$_l_q_allocation]); ! use the quantity from the output
809      0899
810      0900 out_filb [filb$_l_q_allocation] = .blocks; ! Save the value so that we can look at it during the close
811      0901
```

```

812 0902 2 ! Make sure that the record format in the filb is correct
813 0903
814 0904 exch$cmd_fetch_recfmt_implied (.out_filb, .rfp [nam$l_type]+1); ! Pass it the type from the parse
815 0905
816 0906 ! Save the addresses of our routines for this volume and record format.
817 0907
818 0908 out_filb [filb$a_close_routine] = exch$rt11_close_file;
819 0909 out_filb [filb$a_delete_routine] = exch$rt11_delete_file;
820 0910 out_filb [filb$a_get_routine] = 0;
821 0911 out_filb [filb$a_put_routine] = exch$pdput;
822 0912
823 0913 ! Carriage control doesn't mean anything for RT-11 output, tell him we are ignoring
824 0914
825 0915 IF .inp_filb [filb$v_ctl_explicit]
826 0916 OR
827 0917 .out_filb [filb$v_ctl_explicit]
828 0918 THEN
829 0919 sexch_signal (exch$_nocarriage);
830 0920
831 0921 physical = false; ! Assume not using physical transfers
832 0922
833 0923 ! For RT-11 we can treat block transfer mode as fixed 512, physical
834 0924
835 0925 IF .out_filb [filb$b_transfer_mode] EQL filb$k_xfrm_block
836 0926 OR
837 0927 .inp_filb [filb$b_transfer_mode] EQL filb$k_xfrm_block
838 0928 THEN
839 0929 BEGIN
840 0930 physical = true;
841 0931 out_filb [filb$b_rec_format] = filb$k_rfmt_fixed;
842 0932 out_filb [filb$l_fixed_len] = 512;
843 0933 END;
844 0934
845 0935 ! If an explicit record format was given on the input but none on the output, carry the input to the output
846 0936
847 0937 IF .inp_filb [filb$v_rfmt_explicit] ! The input file has explicit format
848 0938 AND (NOT .out_filb [filb$v_rfmt_explicit]) ! but the output has implied record format
849 0939 THEN
850 0940 BEGIN
851 0941 out_filb [filb$b_rec_format] = .inp_filb [filb$b_rec_format];
852 0942 out_filb [filb$l_fixed_len] = .inp_filb [filb$l_fixed_len];
853 0943 END;
854 0944
855 0945 ! In some circumstances we can do block mode I/O rather than record mode
856 0946
857 0947 IF (.inp_filb [filb$b_transfer_mode] EQL filb$k_xfrm_automatic ! Both input and output must be automatic tr
858 0948 AND .out_filb [filb$b_transfer_mode] EQL filb$k_xfrm_automatic)
859 0949 AND
860 0950 ( NOT (.inp_filb [filb$v_rfmt_explicit] ! Both the input and output files must have
861 0951 OR .out_filb [filb$v_rfmt_explicit])) ! implied record formats
862 0952 AND
863 0953 (.inp_namb [namb$b_vol_format] EQL volb$k_vfmt_rt11 ! The input must be RT-11
864 0954 OR .inp_namb [namb$b_vol_format] EQL volb$k_vfmt_dos11) ! or DOS-11
865 0955 THEN
866 0956 BEGIN
867 0957 inp_filb [filb$b_rec_format] = filb$k_rfmt_fixed;
868 0958 inp_filb [filb$l_fixed_len] = 512;
```



```

869 0959 3      out_filb [filb$b_rec_format] = filb$k_rfmt_fixed;
870 0960      out_filb [filb$l_fixed_len] = 512;
871 0961      END;
872 0962
873 0963      ! A block request of zero means grab the largest space on the volume. Let's see if we can determine the exa
874 0964      ! block count we will need.
875 0965
876 0966      IF .blocks EQL 0
877 0967      THEN
878 0968          IF .physical
879 0969              ! Physical works fine
880 0970              OR
881 0971              (
882 0972                  (.inp_filb [filb$b_rec_format] EQL filb$k_rfmt_fixed)
883 0973                  AND
884 0974                  (.out_filb [filb$b_rec_format] EQL filb$k_rfmt_fixed)
885 0975                  AND
886 0976                  (.inp_filb [filb$l_fixed_len] EQL .out_filb [filb$l_fixed_len])
887 0977              )
888 0978          THEN
889 0979              blocks = .inp_filb [filb$l_block_count];
890 0980
891 0981      ! Get some empty area on the volume
892 0982      IF NOT (status = exch$rtacp_find_empty_area (.out_ctx, .blocks, .copy [copy$l_q_start_block]))
893 0983      THEN
894 0984          RETURN .status;
895 0985
896 0986      ! Set the entry up as a tentative file
897 0987
898 0988      ent = .out_ctx [rt11ctx$a_ent_address];
899 0989      ent [rt11ent$b_type_byte] = rt11ent$m_typ_tentative;
900 0990      ! Get the entry pointer into a place where we can use it
901 0991      ! Set the protection attribute of the file. If specified, use that value. Otherwise, if input file is RT-1
902 0992      ! use the attribute of the input.
903 0993
904 0994      IF .copy [copy$v_q_protect_explicit]
905 0995      THEN
906 0996          ! If /PROTECT or /NOPROTECT was explicitly specified
907 0997          ent [rt11ent$v_typ_protected] = .copy [copy$v_q_protect]
908 0998      ELSE
909 0999          BEGIN
910 1000              IF .inp_ctx NEQ 0
911 1001              THEN
912 1002                  IF .inp_ctx [rt11ctx$b_type] EQL exchblk$k_rt11ctx
913 1003                  THEN
914 1004                      ! Input must also be RT-11
915 1005                      ent [rt11ent$v_typ_protected] = .inp_ctx [rt11ctx$v_typ_protected];
916 1006          END;
917 1007
918 1008      ! Get the date into RT11 format
919 1009      exch$rt11_format_current_date (.ent);
920 1010
921 1011      ! Convert the file name to radix 50 and store in the entry
922 1012      exch$util_radix50_from_ascii (.nam_len, .rpf [nam$l_name],
923 1013      rt11ctx$s_exp_name, ent [rt11ent$l_filename]);
924 1014      exch$util_radix50_from_ascii (.typ_len-1, .rpf [nam$l_type]+1,
925 1015      rt11ctx$s_exp_type, ent [rt11ent$w_filetype]);
```

```

926 1016 2
927 1017 2 ! Now force the modified entry to disk
928 1018 2
929 1019 2 exch$rt11_dirseg_put (.volb, .out_ctx [rt11ctx$l_seg_number]);
930 1020 2 CH$MOVE (rt11ent$length, .ent, out_ctx [rt11ctx$l_entry]); ! Put a fresh copy into the context block
931 1021 2
932 1022 2 ! Define a record stream for this file
933 1023 2
934 1024 2 out_ctx [rt11ctx$l_cur_byte] = 0; ! Context is the first byte in
935 1025 2 out_ctx [rt11ctx$l_cur_block] = .out_ctx [rt11ctx$l_start_block]; ! the first block of the file
936 1026 2 out_ctx [rt11ctx$l_eof_block] = .out_ctx [rt11ctx$l_start_block] + .out_ctx [rt11ctx$l_blocks] - 1;
937 1027 2 out_filb [filb$a_record] = 0; ! No valid record or length
938 1028 2 out_filb [filb$l_record_len] = 0;
939 1029 2
940 1030 2 ! Expand the radix-50 filename into the standard ascii text fields
941 1031 2
942 1032 2 exch$rt11_expand_filename (.out_ctx);
943 1033 2
944 1034 2 ! Clear all the flags except the ones we want by writing the masks into the longword
945 1035 2
946 1036 2 out_ctx [rt11ctx$l_flags] = rt11ctx$m_stream_active ! A record stream is currently active
947 1037 2 OR rt11ctx$m_output_file; ! and it is an output file
948 1038 2
949 1039 2 ! Set up the i/o and record buffer
950 1040 2
951 1041 2 IF .out_ctx [rt11ctx$a_buffer] EQL 0
952 1042 2 THEN
953 1043 2 out_ctx [rt11ctx$a_buffer] = exch$util_vm_allocate (ctx$k_buffer_length);
954 1044 2
955 1045 2 ! Set the block pointers to the chunk we are ready to write (i.e. nothing, 'cuz we've done no puts)
956 1046 2
957 1047 2 blocks = MINU (.out_ctx [rt11ctx$l_blocks], ctx$k_buffer_blocks);
958 1048 2 out_ctx [rt11ctx$l_buf_base_block] = .out_ctx [rt11ctx$l_start_block];
959 1049 2 out_ctx [rt11ctx$l_buf_high_block] = .out_ctx [rt11ctx$l_start_block] + blocks - 1;
960 1050 2 out_ctx [rt11ctx$l_high_block_written] = .out_ctx [rt11ctx$l_start_block] - 1;
961 1051 2
962 1052 2 $logic_check (3, (exch$rtacp_verify_directory (.volb)), 189);
963 1053 2
964 1054 2 RETURN true;
965 1055 2 END;
```

.PSECT EXCH\$RT11_PL1T,NOWRT,2

44 41 42 2E 00000 P.AAA: .ASCII \.BAD\ ;

.EXTRN EXCH\$_NOCOPBAD, EXCH\$_NOCOPSAMDEV
.EXTRN EXCH\$_NOCOPDUP, EXCH\$_NOCOPPROT
.EXTRN EXCH\$_NOCOPNODEL
.EXTRN EXCH\$_NOCOPBADDEL
.EXTRN EXCH\$_NOCOPSYSDDEL
.EXTRN EXCH\$_NOCARRIAGE

.PSECT EXCH\$RT11_CODE,NOWRT,2

OFFC 00000

.ENTRY EXCH\$RT11_CREATE_FILE, Save R2,R3,R4,R5,R6,-; 0662

50	00000000G	5E	FE7C	CE	9E	00002	MOVAB	R7,R8,R9,R10,R11	
		EF		04	C1	00007	ADDL3	-388(SP), SP	0712
		59		60	D0	0000F	MOVL	#4, EXCH\$A_GBL, R0	0713
		5A	14	A9	9E	00012	MOVAB	20(R9), R10	
		58	3C	A9	D0	00016	MOVL	60(R9), R8	0715
		56	44	A9	D0	0001A	MOVL	68(R9), R6	0718
		52	035B00FA	8F	D0	0001E	MOVL	#56295674, R2	0725
		51	01AA	8F	3C	00025	MOVZWL	#426, R1	
		50		56	D0	0002A	MOVL	R6, R0	
			00000000G	EF	16	0002D	JSB	EXCH\$UTIL_BLOCK_CHECK	
		52	035B00FA	8F	D0	00033	MOVL	#56295674, R2	0726
		51	0218	8F	3C	0003A	MOVZWL	#536, R1	
		50		58	D0	0003F	MOVL	R8, R0	
			00000000G	EF	16	00042	JSB	EXCH\$UTIL_BLOCK_CHECK	
04		AE	18	A6	D0	00048	MOVL	24(R6), 4(SP)	0727
		52	010A00F7	8F	D0	0004D	MOVL	#17432823, R2	
		51	01EB	8F	3C	00054	MOVZWL	#491, R1	
		50	04	AE	D0	00059	MOVL	4(SP), R0	
			00000000G	EF	16	0005D	JSB	EXCH\$UTIL_BLOCK_CHECK	
		5B	18	A8	D0	00063	MOVL	24(R8), RT1	0728
		52	010A00F7	8F	D0	00067	MOVL	#17432823, R2	
		51	01EC	8F	3C	0006E	MOVZWL	#492, R1	
		50		5B	D0	00073	MOVL	R11, R0	
			00000000G	EF	16	00076	JSB	EXCH\$UTIL_BLOCK_CHECK	
		6E	1C	A6	D0	0007C	MOVL	28(R6), (SP)	0729
		52	041B00F3	8F	D0	00080	MOVL	#68878579, R2	
		51	0213	8F	3C	00087	MOVZWL	#531, R1	
		50		6E	D0	0008C	MOVL	(SP), R0	
			00000000G	EF	16	0008F	JSB	EXCH\$UTIL_BLOCK_CHECK	
50		6E	00000048	8F	C1	00095	ADDL3	#72, (SP), R0	0733
13		60		05	E0	0009D	BBS	#5, (R0), 1\$	
		7E	A6	8F	9A	000A1	MOVZBL	#166, -(SP)	
				01	DD	000A5	PUSHL	#1	
			00000000G	8F	DD	000A7	PUSHL	#EXCH\$ BADLOGIC	
		00		03	FB	000AD	CALLS	#3, LIB\$STOP	
			20	A6	D5	000B4	TSTL	32(R6)	0737
				12	12	000B7	BNEQ	2\$	
				56	DD	000B9	PUSHL	R6	0739
			04	AE	DD	000BB	PUSHL	4(SP)	
				02	FB	000BE	CALLS	#2, EXCH\$UTIL_RT11CTX_ALLOCATE	
		00000000G	EF	50	D0	000C5	MOVL	R0, 32(R6)	
		20	A6	16	11	000C9	BRB	3\$	
		52	008200F4	8F	D0	000CB	MOVL	#8519924, R2	0742
		51	0216	8F	3C	000D2	MOVZWL	#534, R1	
		50	20	A6	D0	000D7	MOVL	32(R6), R0	
			00000000G	EF	16	000DB	JSB	EXCH\$UTIL_BLOCK_CHECK	
		57	20	A6	D0	000E1	MOVL	32(R6), R7	0752
0066	BF	00		00	2C	000E5	MOVC5	#0, (SP), #0, #102, 28(R7)	
		6E		00		000E5			
			1C	A7		000EC			
			24	AE	9F	000EE	PUSHAB	RFP	0759
			5A	A8	9F	000F1	PUSHAB	90(R8)	
			3A	A8	DD	000F4	PUSHL	58(R8)	
			04	AA	DD	000F7	PUSHL	4(R10)	
				6A	3C	000FA	MOVZWL	(R10), -(SP)	
		00000000G	EF	05	FB	000FD	CALLS	#5, EXCH\$CMD_RELATED_FILE_PARSE	
		20	AE	50	D0	00104	MOVL	R0, STATUS	

5A	A6	04	09	20	AE	E8	00108	BLBS	STATUS, 4\$		
			BA		6A	28	0010C	MOV C3	(R10), @4(R10), 90(R6)	0766	
		14	AE	28	032A	31	00112	BRW	36\$	0767	
		14	BE	80	A6	9E	00115	MOVAB	43(R6), 20(SP)	0773	
			SA	5F	8F	8A	0011A	BICB2	#128, @20(SP)		
				70	AE	9A	0011F	MOVZBL	RFP+59, TOT_LEN	0774	
			7E	63	AE	DD	00123	PUSHL	RFP+76	0775	
		00000000G	EF		AE	9A	00126	MOVZBL	RFP+59, -(SP)		
		5F	AE		02	FB	0012A	CALLS	#2, EXCH\$PDP_FILTER_FILENAME		
			50	5F	50	90	00131	MOV B	R0, RFP+59		
			06		AE	9A	00135	MOVZBL	RFP+59, R0	0776	
					50	91	00139	CMPB	R0, #6		
					03	1B	0013C	BLEQU	5\$		
			50		06	D0	0013E	MOVL	#6, R0		
		1C	AE		50	D0	00141	MOVL	R0, NAM_LEN		
		1C	AE		5A	D1	00145	CMPL	TOT_LEN, NAM_LEN	0777	
					05	13	00149	BEQL	6\$		
		14	BE	80	8F	88	0014B	BISB2	#128, @20(SP)	0779	
			SA	60	AE	9A	00150	MOVZBL	RFP+60, TOT_LEN	0781	
	7E	74	AE		01	C1	00154	ADDL3	#1, RFP+80, -(SP)	0782	
			7E	64	AE	9A	00159	MOVZBL	RFP+60, -(SP)		
					6E	D7	0015D	DECL	(SP)		
		00000000G	EF		02	FB	0015F	CALLS	#2, EXCH\$PDP_FILTER_FILENAME		
60	AE		50		01	B1	00166	ADDB3	#1, R0, RFP+80		
			50	60	AE	9A	0016B	MOVZBL	RFP+60, R0	0783	
			04		50	91	0016F	CMPB	R0, #4		
					03	1B	00172	BLEQU	7\$		
			50		04	D0	00174	MOVL	#4, R0		
		18	AE		50	D0	00177	MOVL	R0, TYP_LEN		
		18	AE		5A	D1	0017B	CMPL	TOT_LEN, TYP_LEN	0784	
					05	13	0017F	BEQL	8\$		
		14	BE	80	8F	88	00181	BISB2	#128, @20(SP)	0786	
	5A	1C	AE	18	AE	C1	00186	ADDL3	TYP_LEN, NAM_LEN, TOT_LEN	0788	
	50		6E	00000065	8F	C1	0018C	ADDL3	#10T, (SP), R0	0789	
3A	A6		60		5A	C1	00194	ADDL3	TOT_LEN, (R0), 58(R6)		
		00000100	8F	3A	A6	D1	00199	CMPL	58(R6), #256	0790	
					13	1B	001A1	BLEQU	9\$		
			7E	A4	8F	9A	001A3	MOVZBL	#164, -(SP)		
					01	DD	001A7	PUSHL	#1		
					8F	DD	001A9	PUSHL	#EXCH\$ BADLOGIC		
		00000000G	00		03	FB	001AF	CALLS	#3, LIB\$STOP		
		OC	AE	0100	8F	3C	001B6	MOVZBL	#256, 12(SP)	0791	
		10	AE	5A	A6	9E	001BC	MOVAB	90(R6), 16(SP)	0793	
		08	AE	10	AE	D0	001C1	MOVL	16(SP), 8(SP)		
	7E		6E	00000069	8F	C1	001C6	ADDL3	#105, (SP), -(SP)		
	7E		AE	00000065	8F	C1	001CE	ADDL3	#101, 4(SP), -(SP)		
OC	AE	00	9E		9E	2C	001D7	MOV C5	@(SP)+, @(SP)+, #0, 12(SP), @8(SP)		
				08	BE		001DD				
					38	18	001DF	BGEQ	10\$		
	7E		6E	00000065	8F	C1	001E1	ADDL3	#101, (SP), -(SP)		
		08	AE		9E	C0	001E9	ADDL2	@(SP)+, 8(SP)		
	7E		6E	00000065	8F	C1	001ED	ADDL3	#101, (SP), -(SP)		
		OC	AE		9E	C2	001F5	SUBL2	@(SP)+, 12(SP)		
OC	AE	00	BE	1C	AE	2C	001F9	MOV C5	NAM_LEN, @RFP+76, #0, 12(SP), @8(SP)		
				08	BE		00201				
					14	18	00203	BGEQ	10\$		
		08	AE	1C	AE	C0	00205	ADDL2	NAM_LEN, 8(SP)		

OC	AE	00	0C 74	AE BE	1C 18 08	AE 2C	0020A	SUBL2	NAM_LEN, 12(SP)		
18	AE	00	0000'	CF	74	BE 04	2D 00219	10\$:	CMPC5	#4, P.AAA, #0, TYP_LEN, @RFP+80	0797
		1F	6C	AB		2C 01	12 00223		BNEQ	12\$	
		1A	6C	AB		02 E0	00225		BBS	#1, 108(R11), 11\$	0800
		50	04	AE	0000006C	8F E0	0022A		BBS	#2, 108(R11), 11\$	
		0D		60		01 C1	0022F		ADDL3	#108, 4(SP), R0	0802
		51		AE	0000006C	8F E0	00238		BBS	#1, (R0), 11\$	
		08		61		02 C1	0023C		ADDL3	#108, 4(SP), R1	
				50	00000000G	8F E1	00245		BBC	#2, (R1), 12\$	
						04 D0	00249	11\$:	MOVL	#EXCH\$_NOCOPBAD, R0	0804
						04	00250		RET		
			14	BE	40	8F 8A	00251	12\$:	BICB2	#64, @20(SP)	0811
						5A DD	00256		PUSHL	TOT_LEN	0812
		51		AE	00000065	8F C1	00258		ADDL3	#10T, 4(SP), R1	
		50		56		61 C1	00261		ADDL3	(R1), R6, R0	
					5A	A0 9F	00265		PUSHAB	90(R0)	
						57 DD	00268		PUSHL	R7	
				00000000G	EF	03 FB	0026A		CALLS	#3, EXCH\$RTACP_FIND_FILE	
					03	50 E8	00271		BLBS	R0, 13\$	
						00DB 31	00274		BRW	24\$	
					52	A8 D0	00277	13\$:	MOVL	32(R8), R2	0820
						13 12	0027B		BNEQ	14\$	
					7E	8F 9A	0027D		MOVZBL	#209, -(SP)	
						01 DD	00281		PUSHL	#1	
						8F DD	00283		PUSHL	#EXCH\$_BADLOGIC	
				00000000G	00	03 FB	00289		CALLS	#3, LIB\$STOP	
				F4	8F	A2 91	00290	14\$:	CMPB	10(R2), #244	0824
						1E 12	00295		BNEQ	16\$	
				7E	A7	A2 D1	00297		CMPL	126(R2), 126(R7)	0830
						17 12	0029C		BNEQ	16\$	
		0A		AB		01 E0	0029E		BBS	#1, 108(R11), 15\$	0832
		05		AB		02 E0	002A3		BBS	#2, 108(R11), 15\$	
					30	A9 95	002A8		TSTB	48(R9)	0834
						08 18	002AB		BGEQ	16\$	
					50	8F D0	002AD	15\$:	MOVL	#EXCH\$_NOCOPSAMDEV, R0	0836
						04	002B4		RET		
					43	A7 95	002B5	16\$:	TSTB	67(R7)	0840
						08 13	002B8		BEQL	17\$	
					50	8F D0	002BA		MOVL	#EXCH\$_NOCOPDUP, R0	0842
						04	002C1		RET		
					39	A7 95	002C2	17\$:	TSTB	57(R7)	0844
						08 18	002C5		BGEQ	18\$	
					50	8F D0	002C7		MOVL	#EXCH\$_NOCOPPROT, R0	0846
						04	002CE		RET		
		08		A9		02 E0	002CF	18\$:	BBS	#2, 48(R9), 19\$	0848
				50	00000000G	8F D0	002D4		MOVL	#EXCH\$_NOCOPNODEL, R0	0850
						04	002DB		RET		
			0CAC	8F	3E	A7 B1	002DC	19\$:	CMPW	62(R7), #3244	0852
						08 12	002E2		BNEQ	20\$	
				50	00000000G	8F D0	002E4		MOVL	#EXCH\$_NOCOPBADDEL, R0	0854
						04	002EB		RET		
			7ABB	8F	3E	A7 B1	002EC	20\$:	CMPW	62(R7), #31419	0856
						0D 12	002F2		BNEQ	21\$	
		08		A9		01 E0	002F4		BBS	#1, 49(R9), 21\$	0858

50	00000000G	8F	D0	002F9	MOVL	#EXCH\$NOCOPYSDEL, R0	0860
			04	00300	RET		
		30	A9	95 00301	21\$: TSTB	48(R9)	0864
		47	18	00304	BGEQ	23\$	
		39	A7	95 00306	TSTB	57(R7)	0871
		13	18	00309	BGEQ	22\$	
	7E	83	8F	9A 0030B	MOVZBL	#179, -(SP)	
			01	DD 0030F	PUSHL	#1	
	00000000G		8F	DD 00311	PUSHL	#EXCH\$BADLOGIC	
	00		03	FB 00317	CALLS	#3, LIB\$STOP	
	50	7E	A7	D0 0031E	22\$: MOVL	126(R7), R0	0872
	01	A0	02	90 00322	MOVB	#2, 1(R0)	
		76	A7	DD 00326	PUSHL	118(R7)	0873
		04	AE	DD 00329	PUSHL	4(SP)	
	0000V	CF	02	FB 0032C	CALLS	#2, EXCH\$RT11_DIRSEG_PUT	
1C	30	A9	03	E1 00331	BBC	#3, 48(R9), 24\$	0874
		10	AE	DD 00336	PUSHL	16(SP)	0876
		3A	A6	DD 00339	PUSHL	58(R6)	
			02	DD 0033C	PUSHL	#2	
	00000000G		8F	DD 0033E	PUSHL	#EXCH\$DELETEPREV	
	00		04	FB 00344	CALLS	#4, LIB\$SIGNAL	
		14	BE	40 0034B	BRB	24\$	0864
0066	8F	00	8F	88 0034D	23\$: BISB2	#64, @20(SP)	0882
			00	2C 00352	24\$: MOVCS	#0, (SP), #0, #102, 28(R7)	0889
		1C	A7	00359			
		2D	A8	D5 0035B	TSTL	45(R8)	0894
			06	13 0035E	BEQL	25\$	
		5A	A8	D0 00360	MOVL	45(R8), BLOCKS	0896
			04	11 00364	BRB	26\$	
		5A	A9	D0 00366	25\$: MOVL	36(R9), BLOCKS	0898
	2D	A6	5A	D0 0036A	26\$: MOVL	BLOCKS, 45(R6)	0900
53	74	AE	01	C1 0036E	ADDL3	#1, RFP+80, R3	0904
			53	DD 00373	PUSHL	R3	
			56	DD 00375	PUSHL	R6	
	00000000G	EF	02	FB 00377	CALLS	#2, EXCH\$CMD_FETCH_RECMT IMPLIED	
	4A	A6	CF	9E 0037E	MOVAB	EXCH\$RT11_CLOSE_FILE, 74(R6)	0908
	4E	A6	CF	9E 00384	MOVAB	EXCH\$RT11_DELETE_FILE, 78(R6)	0909
		52	A6	D4 0038A	CLRL	82(R6)	0910
	56	A6	EF	9E 0038D	MOVAB	EXCH\$PDP_PUT, 86(R6)	0911
05	2B	A8	01	E0 00395	BBS	#1, 43(R8), 27\$	0915
0D	14	BE	01	E1 0039A	BBC	#1, @20(SP), 28\$	0917
	00000000G		8F	DD 0039F	27\$: PUSHL	#EXCH\$NOCARRIAGE	0919
	00		01	FB 003A5	CALLS	#1, LIB\$SIGNAL	
			50	D4 003AC	28\$: CLRL	PHYSICAL	0921
		01	A6	91 003AE	CMPB	41(R6), #1	0925
			06	13 003B2	BEQL	29\$	
		01	A8	91 003B4	CMPB	41(R8), #1	0927
			0D	12 003B8	BNEQ	30\$	
	50		01	D0 003BA	29\$: MOVL	#1, PHYSICAL	0930
	28	A6	02	90 003BD	MOVB	#2, 40(R6)	0931
35	A6	0200	8F	3C 003C1	MOVZWL	#512, 53(R6)	0932
	0E	28	A8	E9 003C7	30\$: BLBC	43(R8), 31\$	0937
	0A	14	BE	E8 003CB	BLBS	@20(SP), 31\$	0938
	28	A6	A8	90 003CF	MOVB	40(R8), 40(R6)	0941
35	A6	35	A8	D0 003D4	MOVL	53(R8), 53(R6)	0942
		29	A8	95 003D9	31\$: TSTB	41(R8)	0947
			2D	12 003DC	BNEQ	33\$	

				29	A6	95	003DE	TSTB	41(R6)	0948			
				28	12	003E1	BNEQ	33\$					
		24		28	A8	E8	003E3	BLBS	43(R8), 33\$	0950			
		20		14	BE	E8	003E7	BLBS	220(SP), 33\$	0951			
		03		7A	AB	91	003EB	CMPB	122(R11), #3	0953			
				06	13	003EF	BEQL	32\$					
		01		7A	AB	91	003F1	CMPB	122(R11), #1	0954			
				14	12	003F5	BNEQ	33\$					
	28	A8		02	90	003F7	32\$:	MOVB	#2, 40(R8)	0957			
	35	A8		8F	3C	003FB		MOVZWL	#512, 53(R8)	0958			
	28	A6		02	90	00401		MOVB	#2, 40(R6)	0959			
	35	A6		8F	3C	00405		MOVZWL	#512, 53(R6)	0960			
				5A	D5	0040B	33\$:	TSTL	BLOCKS	0966			
				1A	12	0040D		BNEQ	35\$				
		13		50	E8	0040F		BLBS	PHYSICAL, 34\$	0968			
		02		28	A8	91	00412	CMPB	40(R8), #2	0971			
				11	12	00416		BNEQ	35\$				
		02		28	A6	91	00418	CMPB	40(R6), #2	0973			
				0B	12	0041C		BNEQ	35\$				
	35	A6		35	AB	D1	0041E	CMPL	53(R8), 53(R6)	0975			
				04	12	00423		BNEQ	35\$				
		5A		3E	A8	D0	00425	34\$:	MOVL	62(R8), BLOCKS	0978		
				2C	A9	DD	00429	35\$:	PUSHL	44(R9)	0982		
				0480	8F	BB	0042C		PUSHR	#*M<R7,R10>			
	00000000G	EF		03	FB	00430		CALLS	#3, EXCH\$RTACP_FIND_EMPTY_AREA				
	20	AE		50	D0	00437		MOVL	R0, STATUS				
		05		20	AE	E8	0043B		BLBS	STATUS, 37\$			
		50		20	AE	D0	0043F	36\$:	MOVL	STATUS, R0	0984		
					04	00443		RET					
		52		7E	A7	D0	00444	37\$:	MOVL	126(R7), ENT	0988		
	01	A2			01	90	00448		MOVB	#1, 1(ENT)	0989		
	30	A9			06	E1	0044C		BBC	#6, 48(R9), 38\$	0994		
		01			05	EF	00451		EXTZV	#5, #1, 48(R9), R0	0996		
01	50		30		50	F0	00457		INSV	R0, #7, #1, 1(ENT)			
	A2				19	11	0045D		BRB	39\$			
					50	20	A8	D0	0045F	38\$:	MOVL	32(R8), R0	0999
					13	13	00463		BEQL	39\$			
		F4		8F	0A	A0	91	00465	CMPB	10(R0), #244	1001		
					0C	12	0046A		BNEQ	39\$			
				01	07	EF	0046C		EXTZV	#7, #1, 57(R0), R1	1003		
01	51		39		51	F0	00472		INSV	R1, #7, #1, 1(ENT)	1008		
	A2				52	D0	00478	39\$:	MOVL	ENT, R1			
					0000V	30	0047B		BSBW	EXCH\$RT11_FORMAT_CURRENT_DATE	1013		
				02	A2	9F	0047E		PUSHAB	2(ENT)			
					06	DD	00481		PUSHL	#6			
				78	AE	DD	00483		PUSHL	RFP+76			
				28	AE	DD	00486		PUSHL	NAM_LEN			
		00000000G	EF		04	FB	00489		CALLS	#4, EXCH\$UTIL_RADIX50_FROM_ASCII	1015		
				06	A2	9F	00490		PUSHAB	6(ENT)			
					03	DD	00493		PUSHL	#3			
					53	DD	00495		PUSHL	R3			
	50	24	AE		01	C3	00497		SUBL3	#1, TYP_LEN, R0	1014		
					50	DD	0049C		PUSHL	R0			
		00000000G	EF		04	FB	0049E		CALLS	#4, EXCH\$UTIL_RADIX50_FROM_ASCII	1015		
				76	A7	DD	004A5		PUSHL	118(R7)	1019		
				04	AE	DD	004AB		PUSHL	4(SP)			
		0000V	CF		02	FB	004AB		CALLS	#2, EXCH\$RT11_DIRSEG_PUT			

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_create_file

1 15
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 32
(7)

38	A7	62	24	0E	28	004B0	MOV C3	#14, (ENT), 56(R7)	1020
		52	72	A7	D4	004B5	CLRL	36(R7)	1024
	1C	A7		A7	9E	004B8	MOVAB	114(R7), R2	1025
		50	40	62	D0	004BC	MOVL	(R2), 28(R7)	
		50		A7	3C	004C0	MOVZWL	64(R7), R0	1026
	20	A7	FF	62	C0	004C4	ADDL2	(R2), R0	
			42	A0	9E	004C7	MOVAB	-1(R0), 32(R7)	
				A6	7C	004CC	CLRQ	66(R6)	1028
				57	DD	004CF	PUSHL	R7	1032
	0000V	CF		01	FB	004D1	CALLS	#1, EXCH\$RT11_EXPAND_FILENAME	
	28	A7	18	03	D0	004D6	MOVL	#3, 40(R7)	1037
				A7	D5	004DA	TSTL	24(R7)	1041
		7E	1800	10	12	004DD	BNEQ	40\$	
	00000000G	EF		8F	3C	004DF	MOVZWL	#6144, -(SP)	1043
	18	A7		01	FB	004E4	CALLS	#1, EXCH\$UTIL_VM_ALLOCATE	
		50	40	50	D0	004EB	MOVL	R0, 24(R7)	
		50		A7	3C	004EF	MOVZWL	64(R7), R0	1047
		0C		50	B1	004F3	CMPW	R0, #12	
				03	1B	004F6	BLEQU	41\$	
		50		0C	D0	004F8	MOVL	#12, R0	
		5A		50	D0	004FB	MOVL	R0, BLOCKS	
	2C	A7		62	D0	004FE	MOVL	(R2), 44(R7)	1048
50		62		5A	C1	00502	ADDL3	BLOCKS, (R2), R0	1049
	30	A7	FF	A0	9E	00506	MOVAB	-1(R0), 48(R7)	
34	A7	62		01	C3	0050B	SUBL3	#1, (R2), 52(R7)	1050
		50		01	D0	00510	MOVL	#1, R0	1054
				04	00513		RET		1055

; Routine Size: 1300 bytes. Routine Base: EXCH\$RT11_CODE + 0404

```

967 1056 1 GLOBAL ROUTINE exch$rt11_delete_file (filb : $ref_bblock) = %SBTTL 'exch$rt11_delete_file (filb)'
968 1057 BEGIN
969 1058 ++
970 1059
971 1060 FUNCTIONAL DESCRIPTION:
972 1061
973 1062 Perform RT-11 volume specific delete processing. This is only used to delete output files which we
974 1063 have created, but decide to delete. For example, if the input file were totally unreadable we might
975 1064 delete the output file.
976 1065
977 1066 INPUT/OUTPUT:
978 1067
979 1068 filb - pointer to block describing the file
980 1069
981 1070 IMPLICIT INPUTS:
982 1071
983 1072 none
984 1073
985 1074 OUTPUTS:
986 1075
987 1076 filb - receive info pertaining to the file to be deleted
988 1077
989 1078 IMPLICIT OUTPUTS:
990 1079
991 1080 none
992 1081
993 1082 ROUTINE VALUE:
994 1083
995 1084 true if able to delete the file, false otherwise
996 1085
997 1086 SIDE EFFECTS:
998 1087
999 1088 none
1000 1089 --
1001 1090
1002 1091 $dbgtrc_prefix ('exch$rt11_delete_file> ');
1003 1092
1004 1093 LOCAL
1005 1094 status
1006 1095 ;
1007 1096
1008 1097 BIND
1009 1098 ctx = filb [filb$a_context] : $ref_bblock,
1010 1099 namb = filb [filb$a_assoc_namb] : $ref_bblock,
1011 1100 volb = filb [filb$a_assoc_volb] : $ref_bblock
1012 1101 ;
1013 1102
1014 1103 $debug_print_lit ('entry');
1015 1104
1016 1105 $block_check (2, .filb, filb, 560);
1017 1106 $block_check (2, .ctx, rt11ctx, 561);
1018 1107 $logic_check (3, (.ctx [rt11ctx$y_output_file]), 149);
1019 1108
1020 1109 ; Not much to do, simply leave the file marked as tentative
1021 1110 ;
1022 1111 ctx [rt11ctx$L_flags] = 0;
1023 1112
```


EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_delete_file (filb)

K 15
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 34
(8)

: 1024
: 1025

1113 2 RETURN true;
1114 1 END;

53	04	54	00000000G	EF	9E	00000	.ENTRY	EXCH\$RT11_DELETE_FILE, Save R2,R3,R4	: 1056
		AC		20	C1	00009	MOVAB	EXCH\$UTIL_BLOCK_CHECK, R4	
		52	035B00FA	BF	D0	0000E	ADDL3	#32, FILB, R3	: 1098
		51	0230	BF	3C	00015	MOVL	#56295674, R2	: 1105
		50	04	AC	D0	0001A	MOVZWL	#560, R1	
				64	16	0001E	MOVL	FILB, R0	
		53		63	D0	00020	JSB	EXCH\$UTIL_BLOCK_CHECK	: 1106
		52	008200F4	BF	D0	00023	MOVL	(R3), R3	
		51	0231	BF	3C	0002A	MOVL	#8519924, R2	
		50		53	D0	0002F	MOVZWL	#561, R1	
				64	16	00032	MOVL	R3, R0	
			28	A3	D4	00034	JSB	EXCH\$UTIL_BLOCK_CHECK	: 1111
		50		01	D0	00037	CLRL	40(R3)	: 1113
				04	00	0003A	MOVL	#1, R0	: 1114
							RET		

; Routine Size: 59 bytes, Routine Base: EXCH\$RT11_CODE + 0918

```
1027 1115 1 GLOBAL ROUTINE exch$rt11_dircache_exit_handler (status, %SBTTL 'exch$rt11_dircache_exit_handler (status, vol
1028 1116 1                                     volb : $ref_bblock) : NOVALUE =
1029 1117 1 BEGIN
1030 1118 1 ++
1031 1119 1
1032 1120 1 FUNCTIONAL DESCRIPTION:
1033 1121 1
1034 1122 1     Flush the write cache on the directory.
1035 1123 1
1036 1124 1 INPUTS:
1037 1125 1
1038 1126 1     status - pointer to status code
1039 1127 1     volb - pointer to volb which has been connected to the RT-11 device
1040 1128 1
1041 1129 1 IMPLICIT INPUTS:
1042 1130 1
1043 1131 1     none
1044 1132 1
1045 1133 1 OUTPUTS:
1046 1134 1
1047 1135 1     none
1048 1136 1
1049 1137 1 IMPLICIT OUTPUTS:
1050 1138 1
1051 1139 1     none
1052 1140 1
1053 1141 1 ROUTINE VALUE:
1054 1142 1
1055 1143 1     none
1056 1144 1
1057 1145 1 SIDE EFFECTS:
1058 1146 1
1059 1147 1     any modified directory segments will be written
1060 1148 1
1061 1149 1
1062 1150 1 $dbgtrc_prefix ('rt11_dircache_exit_handler> ');
1063 1151 1
1064 1152 1 BIND
1065 1153 1     fab = volb [volb$a_fab] : $ref_bblock,
1066 1154 1     rab = volb [volb$a_rab] : $ref_bblock
1067 1155 1 ;
1068 1156 1
1069 1157 1 $trace_print_fao ('entry - volb !XL, dircache !XL', .volb, .volb [volb$l_dircache]);
1070 1158 1
1071 1159 1 ! If there are any modified segments signal and flush
1072 1160 1
1073 1161 1 IF .volb [volb$l_dircache] NEQ volb$m_dircache_active
1074 1162 1 THEN
1075 1163 1     BEGIN
1076 1164 1
1077 1165 1         ! Tell we are flushing the directory of a slow device, it might be a while before it finishes
1078 1166 1
1079 1167 1         IF .volb [volb$l_devtype] EQL dt$t_u58          ! If it is any kind of TU58
1080 1168 1         THEN
1081 1169 1             BEGIN
1082 1170 1                 LOCAL
1083 1171 1                 msgvec : VECTOR [5, LONG],
```

```
1084 1172 4      status;
1085 1173 4
1086 1174 4      ! We use the $putmsg service to print this message.  If we signalled it, we could exit the image if
1087 1175 4      ! another signal was active in the catch-all condition handler.  This is extremely likely to happen
1088 1176 4      ! if the control/Y was hit during a command with a /LOG in effect, since the catch-all handler ends
1089 1177 4      ! up printing EXCHANGE log messages.
1090 1178 4
1091 1179 4      msgvec [0] = 4;
1092 1180 4      msgvec [1] = exch$_writecache;
1093 1181 4      msgvec [2] = 2;
1094 1182 4      msgvec [3] = .volb [volb$_vol_ident_len];
1095 1183 4      msgvec [4] = volb [volb$_vol_ident];
1096 1184 5      IF NOT (status = $putmsg (msgvec=msgvec))
1097 1185 4      THEN
1098 1186 4          $exch_signal_stop (.status);
1099 1187 4      END;
1100 1188
1101 1189      ! It is possible that I/O is active (likely if the device is a TU58), so wait for it to complete
1102 1190
1103 1191      IF NOT (status = $wait (rab = .rab))
1104 1192      THEN
1105 1193          exch$_util_file_error (exch$_waiterr, .status, .fab, .rab [rab$_stb]);
1106 1194
1107 1195      ! Call the normal cache stop routine
1108 1196
1109 1197      exch$rt11_dircache_stop (.volb);
1110 1198
1111 1199      END;
1112 1200
1113 1201      RETURN;
1114 1202      END;
```

```
000C 00000
5E      14 C2 00002
53      08 AC D0 00005
01      50 A3 D1 00009
        6B 13 0000D
0E      3C A3 D1 0000F
        34 12 00013
        6E 04 D0 00015
04 AE 00000000G 8F D0 00018
08 AE          02 D0 00020
0C AE          65 A3 D0 00024
10 AE          69 A3 9E 00029
        7E 7C 0002E
        7E D4 00030
        0C AE 9F 00032
00000000G 00 04 FB 00035
        0A 50 E8 0003C
        50 DD 0003F
00000000G 00 01 FB 00041
```

```
.EXTRN EXCH$_WRITECACHE
.EXTRN SY$_PUTMSG, LIB$STOP
.EXTRN SY$_WAIT, EXCH$_WAITERR
```

```
.ENTRY EXCH$RT11_DIRCACHE_EXIT_HANDLER, Save R2,R3 : 1115
SUBL2 #20, SP
MOVL VOLB, R3 : 1153
CML 80(R3), #1 : 1161
BEQL 3$
CML 60(R3), #14 : 1167
BNEQ 1$
MOVL #4, MSGVEC : 1179
MOVL #EXCH$_WRITECACHE, MSGVEC+4 : 1180
MOVL #2, MSGVEC+8 : 1181
MOVL 101(R3), MSGVEC+12 : 1182
MOVAB 105(R3), MSGVEC+16 : 1183
CLRQ -(SP) : 1184
CLRL -(SP)
PUSHAB MSGVEC
CALLS #4, SY$_PUTMSG
BLBS STATUS, 1$
PUSHL STATUS : 1186
CALLS #1, LIB$STOP
```



```
1116 1203 1 GLOBAL ROUTINE exch$rt11_dircache_start (volb : $ref_bblock) : NOVALUE = 7SBTTL 'exch$rt11_dircache_s
1117 1204 2 BEGIN
1118 1205 2 ++
1119 1206 2
1120 1207 2 FUNCTIONAL DESCRIPTION:
1121 1208 2
1122 1209 2 Set up the write cache on the directory.
1123 1210 2
1124 1211 2 INPUTS:
1125 1212 2
1126 1213 2 volb - pointer to volb which has been connected to the RT-11 device
1127 1214 2
1128 1215 2 IMPLICIT INPUTS:
1129 1216 2
1130 1217 2 none
1131 1218 2
1132 1219 2 OUTPUTS:
1133 1220 2
1134 1221 2 none
1135 1222 2
1136 1223 2 IMPLICIT OUTPUTS:
1137 1224 2
1138 1225 2 none
1139 1226 2
1140 1227 2 ROUTINE VALUE:
1141 1228 2
1142 1229 2 none
1143 1230 2
1144 1231 2 SIDE EFFECTS:
1145 1232 2
1146 1233 2 error conditions will be signaled
1147 1234 2 --
1148 1235 2
1149 1236 2 $dbgtrc_prefix ('rt11_dircache_start> ');
1150 1237 2
1151 1238 2 LOCAL
1152 1239 2 status
1153 1240 2 ;
1154 1241 2
1155 1242 2 $block_check (2, .volb, volb, 461);
1156 1243 2 $logic_check (2, (.volb [volb$v_write]), 203); ! We shouldn't get this far if we aren't supposed to write t
1157 1244 2
1158 1245 2 ! If global caching is in effect, ignore this call
1159 1246 2
1160 1247 2 IF .exch$a_gbl [excg$v_q_cache]
1161 1248 2 THEN
1162 1249 2 RETURN;
```

```
1164 1250 2 ! Check some conditions before we proceed
1165 1251 2
1166 1252 2 $trace_print_fao ('entry - volb !XL', .volb);
1167 1253 2 $logic_check (4, (exch$rtacp_verify_directory (.volb)), 204);
1168 1254 2 $logic_check (2, (NOT .volb [volb$v_dircache_active]), 131); ! If it is already on we are confused
1169 1255 2 $logic_check (4, (volb$m_dircache_active EQL 1), 132);
XPRINT: L 1255 2 assumption 132 verified during compilation
1170 1256 2
1171 1257 2 ! Engage directory write caching. Clear all 31 segment flags and activate caching by putting a 1 in the
1172 1258 2 caching longword
1173 1259 2
1174 1260 2 volb [volb$l_dircache] = volb$m_dircache_active;
1175 1261 2
1176 1262 2 ! Declare an exit handler, so that we can flush the cache if the image is run down
1177 1263 2
1178 1264 2 $logic_check (1, (.exch$a_gbl [excg$a_exh_routine] EQL 0), 313); ! There had better not be on
1179 1265 2 exch$a_gbl [excg$a_exh_routine] = exch$rt11_dircache_exit_handler; ! Routine to flush the cache
1180 1266 2 exch$a_gbl [excg$l_exh_arg_count] = 2; ! Status and volb
1181 1267 2 exch$a_gbl [excg$a_exh_status] = exch$a_gbl [excg$l_exh_condvalu]; ! Address to store status
1182 1268 2 exch$a_gbl [excg$a_exh_volb] = .volb; ! Pass address of volb
1183 1269 2
1184 1270 2 IF NOT (status = $dclexh (desblk=exch$a_gbl [excg$r_exit_block]))
1185 1271 2 THEN
1186 1272 2 $exch_signal_stop (.status);
1187 1273 2
1188 1274 2 RETURN;
1189 1275 1 END;
```

```
.EXTRN SYSS$DCLEXH
007C 00000
56 00000000G EF 9E 00002
55 00000000G 8F D0 00009
54 00000000G 00 9E 00010
53 04 AC D0 00017
52 041B00F3 8F D0 0001B
51 01CD 8F 3C 00022
50 53 D0 00027
0B 48 A3 00000000G EF 16 0002A
7E CB 05 E0 00030
01 DD 00039
55 DD 0003B
4F 00 64 03 FB 0003D
0B 50 01 E0 00040 1$:
7E 83 8F 9A 00045
01 DD 0004D
55 DD 0004F
50 64 03 FB 00051 2$:
A3 01 D0 00054
50 30 66 D0 00058
OC 13 0005E
```

```
.ENTRY EXCH$RT11_DIRCACHE_START, Save R2,R3,R4,R5,-; 1203
R6
MOVAB EXCH$a_GBL, R6
MOVL #EXCH$BADLOGIC, R5
MOVAB LIB$STOP, R4
MOVL VOLB, R3 1242
MOVL #68878579, R2
MOVZWL #461, R1
MOVL R3, R0
JSB EXCH$UTIL_BLOCK_CHECK
BBS #5, 72(R3), 1$ 1243
MOVZBL #203, -(SP)
PUSHL #1
PUSHL R5
CALLS #3, LIB$STOP
BBS #1, @EXCH$a_GBL, 4$ 1247
BLBC 80(R3), 2$ 1254
MOVZBL #131, -(SP)
PUSHL #1
PUSHL R5
CALLS #3, LIB$STOP
MOVL #1, 80(R3) 1260
MOVL EXCH$a_GBL, R0 1264
TSTL 48(R0)
BEQL 3$
```


EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dircache_start (volb)

D 16
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 40
(11)

7E	0139	8F	3C	00060	MOVZWL	#313, -(SP)	:
		01	DD	00065	PUSHL	#1	:
		55	DD	00067	PUSHL	R5	:
64		03	FB	00069	CALLS	#3, LIB\$STOP	:
50		66	DD	0006C	MOVL	EXCH\$A_GBL, R0	1265
30	A0	CF	9E	0006F	MOVAB	EXCH\$RT11_DIRCACHE_EXIT_HANDLER, 48(R0)	:
34	A0	02	DD	00075	MOVL	#2, 52(R0)	1266
38	A0	A0	9E	00079	MOVAB	64(R0), 56(R0)	1267
3C	A0	53	DD	0007E	MOVL	R3, 60(R0)	1268
		A0	9F	00082	PUSHAB	44(R0)	1270
00000000G	00	01	FB	00085	CALLS	#1, SYSSDCLEXH	:
	05	50	EB	0008C	BLBS	STATUS, 4\$:
		50	DD	0008F	PUSHL	STATUS	1272
64		01	FB	00091	CALLS	#1, LIB\$STOP	:
		04	00094	4\$:	RET		1275

; Routine Size: 149 bytes, Routine Base: EXCH\$RT11_CODE + 09CE

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dircache_stop (volb)

E 16
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 41
(12)

```
1191 1276 1 GLOBAL ROUTINE exch$rt11_dircache_stop (volb : $ref_bblock) : NOVALUE = %SBTTL 'exch$rt11_dircache_stop (vol
1192 1277 2 BEGIN
1193 1278 3 ++
1194 1279 4
1195 1280 5 FUNCTIONAL DESCRIPTION:
1196 1281 6
1197 1282 7     Clear and flush caches.
1198 1283 8
1199 1284 9 INPUTS:
1200 1285 10
1201 1286 11     volb - pointer to volb which has been connected to the RT-11 device
1202 1287 12
1203 1288 13 IMPLICIT INPUTS:
1204 1289 14
1205 1290 15     none
1206 1291 16
1207 1292 17 OUTPUTS:
1208 1293 18
1209 1294 19     none
1210 1295 20
1211 1296 21 IMPLICIT OUTPUTS:
1212 1297 22
1213 1298 23     none
1214 1299 24
1215 1300 25 ROUTINE VALUE:
1216 1301 26
1217 1302 27     none
1218 1303 28
1219 1304 29 SIDE EFFECTS:
1220 1305 30
1221 1306 31     error conditions will be signaled
1222 1307 32 --
1223 1308 33
1224 1309 34 $dbgtrc_prefix ('rt11_dircache_stop> ');
1225 1310 35
1226 1311 36 $block_check (2, .volb, volb, 457);
```

.ENTRY	EXCH\$RT11_DIRCACHE_STOP, Save R2,R3,R4,R5,-	1276
	R6	
MOVAB	LIB\$STOP, R6	
MOVL	#EXCH\$BADLOGIC, R5	
MOVAB	EXCH\$A-GBL, R4	
MOVL	VOLB, R3	1311
MOVL	#68878579, R2	
MOVZWL	#457, R1	
MOVL	R3, R0	
JSB	EXCH\$UTIL_BLOCK_CHECK	
BBS	#1, @EXCH\$A-GBL, 3\$	1314
BBS	#5, 72(R3), -1\$	1319
MOVZBL	#175, -(SP)	
PUSHL	#1	
PUSHL	R5	
CALLS	#3, LIB\$STOP	
PUSHL	R3	1324
CALLS	#1, EXCH\$RTACP_VERIFY_DIRECTORY	
BLBS	R0, 2\$	
MOVZBL	#178, -(SP)	
PUSHL	#1	
PUSHL	R5	
CALLS	#3, LIB\$STOP	
BICB2	#1, 80(R3)	1328
PUSHL	80(R3)	1329
PUSHL	R3	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dircache_stop (volb)

G 16
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 43
(13)

0000V CF
7E 64
00000000G 00
50

30

02 FB 00065
2C C1 0006A
01 FB 0006E
64 D0 00075
A0 D4 00078
04 0007B 3\$:

CALLS #2, EXCH\$RT11 DIRSEG FLUSH
ADDL3 #44, EXCH\$A_GBL, -(SP)
CALLS #1, SYSSCANEXH
MOVL EXCH\$A_GBL, R0
CLRL 48(R0)
RET

: 1333
: 1334
: 1337

; Routine Size: 124 bytes, Routine Base: EXCH\$RT11_CODE + 0A63

```
1255 1338 1 GLOBAL ROUTINE exch$rt11_dirseg_flush (volb : $ref_bblock,      %SBTTL 'exch$rt11_dirseg_flush (volb, mod)'
1256 1339 1                                     segment_modified : BITVECTOR [32]) =
1257 1340 2 BEGIN
1258 1341 2 ++
1259 1342 2
1260 1343 2 FUNCTIONAL DESCRIPTION:
1261 1344 2
1262 1345 2     Write any directory segments which have been modified.  Whether any actual I/O will occur depends on
1263 1346 2     volb [volb$dircache_active] bit.  If this bit is set, actual I/O will be postponed until flush is
1264 1347 2     with the bit clear.
1265 1348 2
1266 1349 2 INPUTS:
1267 1350 2
1268 1351 2     volb      - pointer to volb which has been connected to the RT-11 device
1269 1352 2     modified_segments - bitvector, set means to write the segment, clear means don't write it
1270 1353 2
1271 1354 2 IMPLICIT INPUTS:
1272 1355 2
1273 1356 2     none
1274 1357 2
1275 1358 2 OUTPUTS:
1276 1359 2
1277 1360 2     none
1278 1361 2
1279 1362 2 IMPLICIT OUTPUTS:
1280 1363 2
1281 1364 2     none
1282 1365 2
1283 1366 2 ROUTINE VALUE:
1284 1367 2
1285 1368 2     true if success, false if failed
1286 1369 2
1287 1370 2 SIDE EFFECTS:
1288 1371 2
1289 1372 2     error conditions will be signaled
1290 1373 2 --
1291 1374 2
1292 1375 2 $dbgtrc_prefix ('rt11_dirseg_flush> ');
1293 1376 2
1294 1377 2 LOCAL
1295 1378 2     seg : $ref_bblock,
1296 1379 2     status
1297 1380 2 ;
1298 1381 2
1299 1382 2 BIND
1300 1383 2     modified_segments = segment_modified      ! map a longword onto the bitvector
1301 1384 2 ;
1302 1385 2
1303 1386 2 $trace_print_fao ('entry - volb !XL, dircache !XL', .volb, .modified_segments);
1304 1387 2
1305 1388 2 $block_check (2, .volb, volb, 532);
1306 1389 2
1307 1390 2 ! Assume that all will go well
1308 1391 2
1309 1392 2 status = true;
1310 1393 2
1311 1394 2 ! A quick exit in case nothing has changed
```

```
1312 1395 2 !
1313 1396 2 ! IF .modified_segments EQL 0
1314 1397 2 ! THEN
1315 1398 2 ! RETURN .status;
1316 1399 2 !
1317 1400 2 ! Find the high segment
1318 1401 2 !
1319 1402 2 ! seg = exch$rt11_dirseg_get (.volb, 1);
1320 1403 2 ! $logic_check (2, (.seg NEQ 0), 210);
1321 1404 2 ! $trace_print_fao ('high segment !UL', .seg [rt11hdr$w_high_seg]);
1322 1405 2 !
1323 1406 2 ! Look at each of the bits, writing those that are set
1324 1407 2 !
1325 1408 2 ! INCRU seg_num FROM 1 TO .seg [rt11hdr$w_high_seg]
1326 1409 2 ! DO
1327 1410 2 ! BEGIN
1328 1411 2 ! $trace_print_fao ('seg num !UL, modified !UL', .seg_num, .segment_modified [.seg_num]);
1329 1412 2 ! IF .segment_modified [.seg_num]
1330 1413 2 ! THEN
1331 1414 2 ! BEGIN
1332 1415 2 ! LOCAL
1333 1416 2 ! temp;
1334 1417 2 !
1335 1418 2 ! temp = exch$rt11_dirseg_put (.volb, .seg_num);
1336 1419 2 ! IF NOT .temp
1337 1420 2 ! THEN
1338 1421 2 ! status = .temp;
1339 1422 2 ! END;
1340 1423 2 ! END;
1341 1424 2 !
1342 1425 2 ! RETURN .status;
1343 1426 2 ! END;
```

52	041B00F3	8F	DO	00002	.ENTRY	EXCH\$RT11_DIRSEG_FLUSH, Save R2,R3,R4	1338
51	0214	8F	3C	00009	MOVL	#68878579, R2	1388
50	04	AC	DO	0000E	MOVZWL	#532, R1	
	00000000G	EF	16	00012	MOVL	VOLB, R0	
54	08	01	DO	00018	JSB	EXCH\$UTIL_BLOCK_CHECK	
		AC	D5	0001B	MOVL	#1, STATUS	1392
		47	13	0001E	TSTL	MODIFIED_SEGMENTS	1396
		01	DD	00020	BEQL	\$S	
	04	AC	DD	00022	PUSHL	#1	1402
0000V	CF	02	FB	00025	PUSHL	VOLB	
52		50	DO	0002A	CALLS	#2, EXCH\$RT11_DIRSEG_GET	
		13	12	0002D	MOVL	R0, SEG	
7E	D2	8F	9A	0002F	BNEQ	1\$	1403
		01	DD	00033	MOVZBL	#210, -(SP)	
	00000000G	8F	DD	00035	PUSHL	#1	
00		03	FB	0003B	PUSHL	#EXCH\$BADLOGIC	
53	04	A2	3C	00042	CALLS	#3, LIB\$STOP	
52		01	DO	00046	MOVZWL	4(SEG), R3	1408
		17	11	00049	MOVL	#1, SEG_NUM	
					BRB	4\$	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dirseg_flush (volb, mod)

J 16
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 46
(14)

10	08	AC	52	E1	0004B	2\$:	BBC	SEG_NUM, SEGMENT_MODIFIED, 3\$	1412
			52	DD	00050		PUSHL	SEG_NUM	1418
			AC	DD	00052		PUSHL	VOLB	1419
	0000V	CF	02	FB	00055		CALLS	#2, EXCH\$RT11_DIRSEG_PUT	1421
		03	50	E8	0005A		BLBS	TEMP, 3\$	1408
		54	50	D0	0005D		MOVL	TEMP, STATUS	1425
			52	D6	00060	3\$:	INCL	SEG_NUM	1426
		53	52	D1	00062	4\$:	CMPL	SEG_NUM, R3	
			E4	1B	00065		BLEQU	2\$	
		50	54	D0	00067	5\$:	MOVL	STATUS, R0	
			04	0006A			RET		

; Routine Size: 107 bytes, Routine Base: EXCH\$RT11_CODE + 0ADF


```
1345 1427 1 GLOBAL ROUTINE exch$rt11_dirseg_get (volb : $ref_bblock, number) = %SBTTL 'exch$rt11_dirseg_get (volb)'
1346 1428 BEGIN
1347 1429 ++
1348 1430
1349 1431 FUNCTIONAL DESCRIPTION:
1350 1432
1351 1433     Return a pointer to the requested directory segment
1352 1434
1353 1435 INPUTS:
1354 1436
1355 1437     volb - pointer to volb which has been connected to the RT-11 device
1356 1438     number - directory segment number in the range 1-31
1357 1439
1358 1440 IMPLICIT INPUTS:
1359 1441
1360 1442     none
1361 1443
1362 1444 OUTPUTS:
1363 1445
1364 1446     none
1365 1447
1366 1448 IMPLICIT OUTPUTS:
1367 1449
1368 1450     none
1369 1451
1370 1452 ROUTINE VALUE:
1371 1453
1372 1454     address of segment, or 0 if any error
1373 1455
1374 1456 SIDE EFFECTS:
1375 1457
1376 1458     error conditions will be signaled
1377 1459
1378 1460 --
1379 1461 $dbgtrc_prefix ('rt11_dirseg_get> ');
1380 1462
1381 1463 LOCAL
1382 1464     rrv : $ref_bblock,           ! a pointer to the rt11 volb extension
1383 1465     rot : $ref_bblock,           ! a pointer to the root directory segment
1384 1466     seg : $ref_bblock           ! a pointer to the desired segment
1385 1467 ;
1386 1468
1387 1469 $block_check (2, .volb, volb, 453);
1388 1470 $trace_print_fao (' entry - volb !XL, seg !2UL, dircache !XL', .volb, .number, .volb [volb$l_dircache]);
1389 1471
1390 1472 ! Get the pointer to our volb extension and to the root segment
1391 1473
1392 1474 rrv = .volb [volb$a_vfmt_specific];
1393 1475 $block_check (2, .rrv, rrv, 454);
1394 1476 rot = rrv [rt11$t_block_0] + (512 * rt11$k_root_block);
1395 1477
1396 1478 ! We assume that the directory (thru high_seg not num_segs) is present in memory
1397 1479
1398 1480 $logic_check (2, .rrv [rt11$v_dir_present], 124);
1399 1481
1400 1482 ! Check the consistency of the root segment. The following tests are order-dependent. The BLISS optimizer
1401 1483 ! fold them all together into a single signal and return, but if we had all the tests inside a single IF sta
```

```
1402 1484 2 ! the optimizer might have executed them in any order it felt like.
1403 1485
1404 1486 IF (.rot [rt11hdr$w_num_segs] EQL 0)
1405 1487 OR
1406 1488 (.rot [rt11hdr$w_num_segs] GTRU 31)
1407 1489 THEN
1408 1490 BEGIN
1409 1491 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1410 1492 RETURN 0;
1411 1493 END;
1412 1494 IF (.rot [rt11hdr$w_high_seg] EQL 0)
1413 1495 OR
1414 1496 (.rot [rt11hdr$w_high_seg] GTRU .rot [rt11hdr$w_num_segs])
1415 1497 THEN
1416 1498 BEGIN
1417 1499 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1418 1500 RETURN 0;
1419 1501 END;
1420 1502 IF (.rot [rt11hdr$w_next_seg] GTRU .rot [rt11hdr$w_high_seg])
1421 1503 THEN
1422 1504 BEGIN
1423 1505 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1424 1506 RETURN 0;
1425 1507 END;
1426 1508
1427 1509 ! The RT-11 Version 4 DUP objects if more than 119 extra words are specified in an initialize (/Z:120. fails
1428 1510 ! Since strange things can happen (like directories which can hold < 1 file) if this number is large, we are
1429 1511 ! going to complain if it exceeds this number too.
1430 1512
1431 1513 IF (.rot [rt11hdr$w_extra_bytes] GTRU 238)
1432 1514 THEN
1433 1515 BEGIN
1434 1516 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1435 1517 RETURN 0;
1436 1518 END;
1437 1519 IF ((.rot [rt11hdr$w_extra_bytes] AND 1) NEQ 0) ! It can't be odd either
1438 1520 THEN
1439 1521 BEGIN
1440 1522 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1441 1523 RETURN 0;
1442 1524 END;
1443 1525
1444 1526 ! Do a bounds check on the requested segment
1445 1527
1446 1528 IF (.number EQL 0)
1447 1529 OR
1448 1530 (.number GTRU .rot [rt11hdr$w_high_seg])
1449 1531 THEN
1450 1532 BEGIN
1451 1533 $exch_signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1452 1534 RETURN 0;
1453 1535 END;
1454 1536
1455 1537 ! Looks good, now compute the desired segment as an offset from the root
1456 1538
1457 1539 seg = .rot + ((.number-1) * rt11$k_dirseglen);
1458 1540
```

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dirseg_get (volb)

M 16
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 49
(15)

```

1459 1541 2 ! Now perform some consistency checks on the segment header
1460 1542 2
1461 1543 2 IF (.seg [rt11hdr$w_num_segs] NEQ .rot [rt11hdr$w_num_segs])
1462 1544 2 OR
1463 1545 2 (.seg [rt11hdr$w_next_seg] GTRU .rot [rt11hdr$w_high_seg])
1464 1546 2 OR
1465 1547 2 (.seg [rt11hdr$w_extra_bytes] NEQ .rot [rt11hdr$w_extra_bytes])
1466 1548 2 THEN
1467 1549 2 BEGIN
1468 1550 2 $exch signal (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1469 1551 2 RETURN 0;
1470 1552 2 END;
1471 1553 2
1472 1554 2 RETURN .seg;
1473 1555 2 END;

```

```

                                .EXTRN EXCH$RT11_BADDIRECT
                                .ENTRY EXCH$RT11_DIRSEG_GET, Save R2,R3,R4,R5,R6
                                MOVAB EXCH$UTIL_BLOCK_CHECK, R6
                                MOVL VOLB, R3
                                MOVL #68878579, R2
                                MOVZWL #453, R1
                                MOVL R3, R0
                                JSB EXCH$UTIL_BLOCK_CHECK
                                MOVL 84(R3), RTV
                                MOVL #-2012348171, R2
                                MOVZWL #454, R1
                                MOVL RTV, R0
                                JSB EXCH$UTIL_BLOCK_CHECK
                                MOVAB 3086(R4), ROT
                                BLBS 12(RTV), 1$
                                MOVZBL #124, -(SP)
                                PUSHL #1
                                PUSHL #EXCH$BADLOGIC
                                CALLS #3, LIB$STOP
                                TSTW (ROT)
                                BEQL 2$
                                CMPW (ROT), #31
                                BGTRU 2$
                                MOVZWL 4(ROT), R5
                                BEQL 2$
                                CMPW (ROT), R5
                                BLSSU 2$
                                CMPW 2(ROT), R5
                                BGTRU 2$
                                CMPW 6(ROT), #238
                                BGTRU 2$
                                BLBS 6(ROT), 2$
                                MOVL NUMBER, R4
                                BEQL 2$
                                CMPL R4, R5
                                BGTRU 2$
                                ASHL #10, R4, R0
                                MOVAB -1024(R0)[ROT], SEG

```

56	00000000G	EF	9E	00000	
53	04	AC	D0	00009	
52	041B00F3	8F	D0	0000D	
51	01C5	8F	3C	00014	
50		53	D0	00019	
		66	16	0001C	
54	54	A3	D0	0001E	
52	880E00F5	8F	D0	00022	
51	01C6	8F	3C	00029	
50		54	D0	0002E	
		66	16	00031	
52	0C0E	C4	9E	00033	
13	0C	A4	E8	00038	
7E	7C	8F	9A	0003C	
		01	DD	00040	
	00000000G	8F	DD	00042	
	00	03	FB	00048	
		62	B5	0004F	1\$:
		49	13	00051	
1F		62	B1	00053	
		44	1A	00056	
55	04	A2	3C	00058	
		3E	13	0005C	
55		62	B1	0005E	
		39	1F	00061	
55	02	A2	B1	00063	
		33	1A	00067	
00EE	8F	06	A2	B1	00069
		2B	1A	0006F	
27	06	A2	E8	00071	
54	08	AC	D0	00075	
		21	13	00079	
55		54	D1	0007B	
		1C	1A	0007E	
50	54	0A	78	00080	
	54	FC00	C042	9E	00084

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dirseg_get (volb)

B 1
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 50
(15)

62		64	B1	0008A	CMPW	(SEG), (ROT)	: 1543
		0D	12	0008D	BNEQ	2\$: 1545
55	02	A4	B1	0008F	CMPW	2(SEG), R5	: 1547
		07	1A	00093	BGTRU	2\$: 1550
06	A2	06	A4	B1	00095	CMPW	6(SEG), 6(ROT)
			17	13	0009A	BEQL	3\$
		69	A3	9F	0009C	PUSHAB	105(R3)
		65	A3	DD	0009F	PUSHL	101(R3)
			02	DD	000A2	PUSHL	#2
00000000G	00	00000000G	8F	DD	000A4	PUSHL	#EXCH\$ RT11_BADDIRECT
			04	FB	000AA	CALLS	#4, LIB\$SIGNAL
			04	11	000B1	BRB	4\$
	50		54	D0	000B3	MOVL	SEG, R0
				04	000B6	RET	
		50	D4	000B7	CLRL	R0	: 1555
			04	000B9	RET		: 1555

; Routine Size: 186 bytes, Routine Base: EXCH\$RT11_CODE + 0B4A


```
1475 1556 1 GLOBAL ROUTINE exch$rt11_dirseg_get_nochk (volb : $ref_bblock, number) : jsb_r1r2 = %SBTTL 'exch$rt11_dirseg_get_nochk'
1476 1557 2 BEGIN
1477 1558 3 ++
1478 1559 4
1479 1560 5 FUNCTIONAL DESCRIPTION:
1480 1561 6
1481 1562 7     Return a pointer to the requested directory segment without any checking
1482 1563 8
1483 1564 9 INPUTS:
1484 1565 10
1485 1566 11     volb - pointer to volb which has been connected to the RT-11 device
1486 1567 12     number - directory segment number in the range 1-31
1487 1568 13
1488 1569 14 IMPLICIT INPUTS:
1489 1570 15
1490 1571 16     none
1491 1572 17
1492 1573 18 OUTPUTS:
1493 1574 19
1494 1575 20     none
1495 1576 21
1496 1577 22 IMPLICIT OUTPUTS:
1497 1578 23
1498 1579 24     none
1499 1580 25
1500 1581 26 ROUTINE VALUE:
1501 1582 27
1502 1583 28     address of segment, or 0 if any error
1503 1584 29
1504 1585 30 SIDE EFFECTS:
1505 1586 31
1506 1587 32     error conditions will be signaled
1507 1588 33
1508 1589 34 --
1509 1590 35 $dbgtrc_prefix ('rt11_dirseg_get_nochk> ');
1510 1591 36
1511 1592 37 BIND
1512 1593 38     rrv = volb [volb$a_vfmt_specific] : $ref_bblock ! a pointer to the rt11 volb extension
1513 1594 39 ;
1514 1595 40
1515 1596 41 $debug_print_lit ('entry');
1516 1597 42
1517 1598 43 ! Get the pointer to our volb extension and to the root segment, then compute the
1518 1599 44
1519 1600 45 RETURN rrv [rt11$t_block_0] + (512 * rt11$k_root_block) + ((.number-1) * rt11$k_dirseglen);
1520 1601 46 END;
```

52	52	0A	78	00000	EXCH\$RT11 DIRSEG GET_NOCHK::		
	52				ASHL #10, R2, R2	1600	
	52	54	A1	C0	00004	ADDL2 84(VOLB), R2	
	52	080E	C2	9E	00008	MOVAB 2062(R2), R2	
	50		52	D0	0000D	MOVL R2, R0	
				05	00010	RSB	1601

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dirseg_get_nochk

D 1
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 52
(16)

; Routine Size: 17 bytes, Routine Base: EXCH\$RT11_CODE + 0C04

```
1522 1602 1 GLOBAL ROUTINE exch$rt11_dirseg_put (volb : $ref_bblock, number) = %SBTTL 'exch$rt11_dirseg_put (volb,
1523 1603 2 BEGIN
1524 1604 2 ++
1525 1605 2
1526 1606 2 FUNCTIONAL DESCRIPTION:
1527 1607 2
1528 1608 2 Write a directory segment back to disk
1529 1609 2
1530 1610 2 INPUTS:
1531 1611 2
1532 1612 2 volb - pointer to volb which has been connected to the RT-11 device
1533 1613 2 number - directory segment number in the range 1-31
1534 1614 2
1535 1615 2 IMPLICIT INPUTS:
1536 1616 2
1537 1617 2 none
1538 1618 2
1539 1619 2 OUTPUTS:
1540 1620 2
1541 1621 2 none
1542 1622 2
1543 1623 2 IMPLICIT OUTPUTS:
1544 1624 2
1545 1625 2 none
1546 1626 2
1547 1627 2 ROUTINE VALUE:
1548 1628 2
1549 1629 2 true if success, error code if problem arose
1550 1630 2
1551 1631 2 SIDE EFFECTS:
1552 1632 2
1553 1633 2 error conditions will be signaled
1554 1634 2 --
1555 1635 2
1556 1636 2 $dbgtrc_prefix ('rt11_dirseg_put> ');
1557 1637 2
1558 1638 2 LOCAL
1559 1639 2 blk, ! pbn of block to write
1560 1640 2 rtv : $ref_bblock, ! a pointer to the rt11 volb extension
1561 1641 2 rot : $ref_bblock, ! a pointer to the root directory segment
1562 1642 2 seg : $ref_bblock, ! a pointer to the desired segment
1563 1643 2 status
1564 1644 2 :
1565 1645 2
1566 1646 2 $block_check (2, .volb, volb, 529);
1567 1647 2 $trace_print_fao ('* entry = volb !XL, seg !2UL, dircache !XL', .volb, .number, .volb [volb$l_dircache]);
1568 1648 2 $logic_check (2, (.volb [volb$v_write]), 146); ! We shouldn't get this far if we aren't supposed to write t
1569 1649 2
1570 1650 2 ! Get the pointer to our volb extension and to the root segment
1571 1651 2
1572 1652 2 rtv = .volb [volb$a_vfmt_specific];
1573 1653 2 $block_check (2, .rtv, rt11, 528);
1574 1654 2 rot = rtv [rt11$t_block_0] + (512 * rt11$k_root_block);
1575 1655 2
1576 1656 2 ! We assume that the directory (thru high_seg not num_segs) is present in memory
1577 1657 2
1578 1658 2 $logic_check (2, .rtv [rt11$v_dir_present], 142);
```

```
1579 1659 2
1580 1660 2 ! Do a bounds check on the requested segment
1581 1661 2
1582 1662 2 IF (.number EQL 0)
1583 1663 2 OR
1584 1664 2 (.number GTRU .rot [rt11hdr$w_high_seg])
1585 1665 2 THEN
1586 1666 2 $exch_signal_return (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident]);
1587 1667 2
1588 1668 2 ! Segment number is valid, if caching is on we just set a bit in the cache bitvector !!! interim cache st
1589 1669 2
1590 1670 2 IF .volb [volb$v_dircache_active]
1591 1671 2 THEN
1592 1672 2 BEGIN
1593 1673 2 BIND
1594 1674 2 segbit = volb [volb$l_dircache] : BITVECTOR [32];
1595 1675 2 segbit [.number] = true;
1596 1676 2 status = true;
1597 1677 2 END
1598 1678 2
1599 1679 2 ! Caching not active, write the segment immediately
1600 1680 2
1601 1681 2 ELSE
1602 1682 2 BEGIN
1603 1683 2
1604 1684 2 ! Looks good, now find the address of the desired segment, and the block number
1605 1685 2
1606 1686 2 seg = .rot + ((.number-1) * rt11$k_dirseglen);
1607 1687 2 blk = 2*(.number-1) + rt11$k_root_block;
1608 1688 2
1609 1689 2 ! Now perform some consistency checks on the segment header
1610 1690 2
1611 1691 2 IF (.seg [rt11hdr$w_num_segs] NEQ .rot [rt11hdr$w_num_segs])
1612 1692 2 OR
1613 1693 2 (.seg [rt11hdr$w_next_seg] GTRU .rot [rt11hdr$w_high_seg])
1614 1694 2 THEN
1615 1695 2 $exch_signal_return (exch$rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident])
1616 1696 2
1617 1697 2 ! Write the directory segment back to the disk
1618 1698 2
1619 1699 2 status = exch$io_rt11_write (.volb, .blk, 2, .seg);
1620 1700 2
1621 1701 2 ! If there was an error, we should give them extra warning that quick action can save the file
1622 1702 2
1623 1703 2 IF (NOT .status)
1624 1704 2 THEN
1625 1705 2 $exch_signal (exch$dire_error);
1626 1706 2
1627 1707 2 ! Following expression would print additional information to direct the user as to recovery
1628 1708 2 procedures, so that he could save all the information in the volume by using the correct
1629 1709 2 copy of the directory which is still in memory.
1630 1710 2
1631 1711 2 IF .exch$a_gbl [excg$v_foreign_command] ! If single command, no hope of recovery, sigh...
1632 1712 2 THEN
1633 1713 2 $exch_signal (exch$dire_error)
1634 1714 2 ELSE
1635 1715 2 $exch_signal (exch$dire_error, 0, exch$recover);
```



```
: 1636
: 1637
: 1638
: 1639
: 1640
```

```
1716 3
1717 2 END;
1718 2
1719 2 RETURN .status;
1720 1 END;
```

				OFFC 00000		
				5B 00000000G	8F DO 00002	
				5A 00000000G	00 9E 00009	
				59 00000000G	8F DO 00010	
				58 00000000G	00 9E 00017	
				54 04	AC DO 0001E	
				52 041B00F3	8F DO 00022	
				51 0211	8F 3C 00029	
				50 00000000G	54 DO 0002E	
					EF 16 00031	
OB	48			A4 92	05 E0 00037	
				7E	8F 9A 0003C	
					01 DD 00040	
					59 DD 00042	
				6A 54	03 FB 00044	
				55 880E00F5	A4 DO 00047 1\$:	
				52 0210	8F DO 0004B	
				51	8F 3C 00052	
				50 00000000G	55 DO 00057	
					EF 16 0005A	
				53 0C0E	C5 9E 00060	
				OB 0C	A5 E8 00065	
				7E 8E	8F 9A 00069	
					01 DD 0006D	
					59 DD 0006F	
				6A 08	03 FB 00071	
				56	AC DO 00074 2\$:	
					08 13 00078	
56	04	A3		10	00 ED 0007A	
					14 1E 00080	
				52	5B DO 00082 3\$:	
					69 A4 9F 00085	
					65 A4 DD 00088	
					02 DD 0008B	
					52 DD 0008D	
				68	04 FB 0008F	
				50	52 DO 00092	
					04 00095	
				0A 50	A4 E9 00096 4\$:	
	00	50		A4	56 E2 0009A	
				57	01 DO 0009F 5\$:	
					4F 11 000A2	
	52			56	0A 78 000A4 6\$:	
				55 FC00 C243	9E 000A8	
	52			56	01 78 000AE	
				52	04 C0 000B2	

		.EXTRN EXCH\$_DIRE_ERROR	
		.ENTRY EXCH\$RT11_DIRSEG_PUT, Save R2,R3,R4,R5,R6,-	1602
		R7,R8,R9,R10,R11	
		MOVL #EXCH\$ RT11_BADDIRECT, R11	
		MOVAB LIB\$STOP, R10	
		MOVL #EXCH\$ BADLOGIC, R9	
		MOVAB LIB\$SIGNAL, R8	
		MOVL VOLB, R4	1646
		MOVL #68878579, R2	
		MOVZWL #529, R1	
		MOVL R4, R0	
		JSB EXCH\$UTIL_BLOCK_CHECK	
		BBS #5, 72(R4), 1\$	1648
		MOVZBL #146, -(SP)	
		PUSHL #1	
		PUSHL R9	
		CALLS #3, LIB\$STOP	
		MOVL 84(R4), RTV	1652
		MOVL #-2012348171, R2	1653
		MOVZWL #528, R1	
		MOVL RTV, R0	
		JSB EXCH\$UTIL_BLOCK_CHECK	
		MOVAB 3086(R5), -ROT	1654
		BLBS 12(RTV), 2\$	1658
		MOVZBL #142, -(SP)	
		PUSHL #1	
		PUSHL R9	
		CALLS #3, LIB\$STOP	
		MOVL NUMBER, R6	1662
		BEQL 3\$	
		CMPZV #0, #16, 4(ROT), R6	1664
		BGEQU 4\$	
		MOVL R11, TEMP	1666
		PUSHAB 105(R4)	
		PUSHL 101(R4)	
		PUSHL #2	
		PUSHL TEMP	
		CALLS #4, LIB\$SIGNAL	
		MOVL TEMP, R0	
		RET	
		BLBC 80(R4), 6\$	1670
		BBSS R6, 80(R4), 5\$	1675
		MOVL #1, STATUS	1676
		BRB 9\$	1670
		ASHL #10, R6, R2	1686
		MOVAB -1024(R2)[ROT], SEG	
		ASHL #1, R6, BLK	1687
		ADDL2 #4, BLK	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_dirseg_put (volb, number)

H 1
16-Sep-1984 01:14:37 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:29:07 [EXCHNG.SRC]EXCRT11.B32;1

Page 56
(17)

	63		65	B1	000B5	CMPW	(SEG), (ROT)	1691
			07	12	000B8	BNEQ	7\$	
04	A3	02	A5	B1	000BA	CMPW	2(SEG), 4(ROT)	1693
	56		14	1B	000BF	BLEQU	8\$	
		69	5B	D0	000C1	MOVL	R11, TEMP	1695
		65	A4	9F	000C4	PUSHAB	105(R4)	
			A4	DD	000C7	PUSHL	101(R4)	
			02	DD	000CA	PUSHL	#2	
			56	DD	000CC	PUSHL	TEMP	
68			04	FB	000CE	CALLS	#4, LIB\$SIGNAL	
50			56	D0	000D1	MOVL	TEMP, R0	
				04	000D4	RET		
			55	DD	000D5	PUSHL	SEG	1699
			02	DD	000D7	PUSHL	#2	
			52	DD	000D9	PUSHL	BLK	
			54	DD	000DB	PUSHL	R4	
00000000G	EF		04	FB	000DD	CALLS	#4, EXCH\$IO_RT11_WRITE	
	57		50	D0	000E4	MOVL	R0, STATUS	
	09		57	E8	000E7	BLBS	STATUS, 9\$	1703
		00000000G	8F	DD	000EA	PUSHL	#EXCH\$ DIRE ERROR	1705
68			01	FB	000F0	CALLS	#1, LIB\$SIGNAL	
50			57	D0	000F3	MOVL	STATUS, R0	1719
			04	000F6	RET			1720

; Routine Size: 247 bytes, Routine Base: EXCH\$RT11_CODE + 0C15

```
1642 1721 1 GLOBAL ROUTINE exch$rt11_expand_filename (ctx : $ref_bblock) = %SBTTL 'exch$rt11_expand_filename (ctx)'
1643 1722 2 BEGIN
1644 1723 3 ++
1645 1724 4
1646 1725 5 FUNCTIONAL DESCRIPTION:
1647 1726 6
1648 1727 7     Convert the information in a directory entry to ascii text. This involves changing the RADIX-50
1649 1728 8     filename to ASCII and converting the date to ASCII.
1650 1729 9
1651 1730 10 INPUTS:
1652 1731 11
1653 1732 12     ctx - pointer to an rt11ctx structure which contains a copy of the directory entry
1654 1733 13
1655 1734 14 IMPLICIT INPUTS:
1656 1735 15
1657 1736 16     none
1658 1737 17
1659 1738 18 OUTPUTS:
1660 1739 19
1661 1740 20     ctx - receives ASCII filename info
1662 1741 21
1663 1742 22 IMPLICIT OUTPUTS:
1664 1743 23
1665 1744 24     none
1666 1745 25
1667 1746 26 ROUTINE VALUE:
1668 1747 27
1669 1748 28     success or failure if the entry is invalid
1670 1749 29
1671 1750 30 SIDE EFFECTS:
1672 1751 31
1673 1752 32     error conditions will be signaled
1674 1753 33 --
1675 1754 34 $dbgtrc_prefix ('rt11_expand_filename> ');
1676 1755 35
1677 1756 36 OWN
1678 1757 37     ! Read-only own
1679 1758 38     months : VECTOR [13, LONG] INITIAL
1680 1759 39         ('Jan-', 'Feb-', 'Mar-', 'Apr-', 'May-', 'Jun-', 'Jul-', 'Aug-', 'Sep-', 'Oct-', 'Nov-', 'Dec-', '***-')
1681 1760 40     ;
1682 1761 41
1683 1762 42 LOCAL
1684 1763 43     year,
1685 1764 44     mon,
1686 1765 45     day,
1687 1766 46     date_desc : VECTOR [2, LONG],
1688 1767 47     status,
1689 1768 48     ch
1690 1769 49     ;
1691 1770 50
1692 1771 51 $block_check (2, .ctx, rt11ctx, 452);
1693 1772 52
1694 1773 53 ! Convert the file name from 2 Radix-50 words to 'rt11ctx$s_exp_name' ASCII characters, type from 1 R50 word
1695 1774 54 ! 'RT11ctx$s_exp_type' chars
1696 1775 55
1697 1776 56 exch$util_radix50_to_ascii (rt11ctx$s_exp_name, ctx [rt11ctx$L_filename], ctx [rt11ctx$t_exp_name]);
1698 1777 57 ch = CH$FIND_CH (rt11ctx$s_exp_name, ctx [rt11ctx$t_exp_name], '');
```

```
1699 1778 2 ctx [rt11ctx$exp_name_len] = (IF .ch EQL 0 THEN rt11ctx$exp_name ELSE .ch - ctx [rt11ctx$exp_name]);
1700 1779
1701 1780 2 exch$util_radix50 to ascii (rt11ctx$exp_type, ctx [rt11ctx$exp_filetype], ctx [rt11ctx$exp_type]);
1702 1781 2 ch = CH$FIND CH (rt11ctx$exp_type, ctx [rt11ctx$exp_type], '.');
1703 1782 2 ctx [rt11ctx$exp_type_len] = (IF .ch EQL 0 THEN rt11ctx$exp_type ELSE .ch - ctx [rt11ctx$exp_type]);
1704 1783
1705 1784 2 ! If file is protected, set the P
1706 1785
1707 1786 2 IF .ctx [rt11ctx$typ_protected]
1708 1787 2 THEN
1709 1788 2     CH$MOVE (2, UPLIT BYTE ('p-'), ctx [rt11ctx$exp_protected])
1710 1789 2 ELSE
1711 1790 2     CH$MOVE (2, UPLIT BYTE (' '), ctx [rt11ctx$exp_protected]);
1712 1791
1713 1792 2 ! Create a filename in the standard, non-embedded blank format by concatenating the name, a "." and the type
1714 1793 2
1715 1794 2 ctx [rt11ctx$exp_fullname_len] = .ctx [rt11ctx$exp_name_len] + 1 + .ctx [rt11ctx$exp_type_len];
1716 1795 2 CH$COPY (.ctx [rt11ctx$exp_name_len], ctx [rt11ctx$exp_name], ! the file name
1717 1796 2     1, UPLIT BYTE ('.'), ! the "." separator
1718 1797 2     .ctx [rt11ctx$exp_type_len], ctx [rt11ctx$exp_type], ! the file type
1719 1798 2     'C ', rt11ctx$exp_fullname, ctx [rt11ctx$exp_fullname]); ! the blank-padded result
1720 1799
1721 1800 2 ! Create an ASCII representation of the date
1722 1801 2
1723 1802 2 IF .ctx [rt11ctx$exp_date] EQL 0
1724 1803 2 THEN
1725 1804 2     CH$MOVE (rt11ctx$exp_date, UPLIT BYTE (' < nodate >'), ctx [rt11ctx$exp_date])
1726 1805 2 ELSE
1727 1806 2     BEGIN
1728 1807 2         year = 1972 + .ctx [rt11ctx$year]; ! 1972 is stored as zero
1729 1808 2         day = .ctx [rt11ctx$day]; ! day is stored 1-31
1730 1809 2         mon = .ctx [rt11ctx$month] - 1; ! month is 1-12, adjust for vector index
1731 1810 2         IF .mon GTRU 12 THEN mon = 12; ! point bad months at '***-'
1732 1811 2         date_desc [0] = rt11ctx$exp_date; ! set up a descriptor for FAO
1733 1812 2         date_desc [1] = ctx [rt11ctx$exp_date];
1734 1813 2         IF NOT (status = $fao (%ASCID '12UC-1AF14UL', 0, date_desc, .day, 4, months [.mon], .year))
1735 1814 2         THEN
1736 1815 2             $exch_signal_stop (.status);
1737 1816 2         END;
1738 1817
1739 1818 2 %IF switch_debug
1740 1819 2 %THEN
1741 1820 2     BEGIN
1742 1821 2     LOCAL
1743 1822 2         ent_typ;
1744 1823 2     BIND
1745 1824 2         a = ctx [rt11ctx$entry] : VECTOR [, WORD];
1746 1825 2         ! Show the entry type
1747 1826 2         ent_typ = (CASE .ctx [rt11ctx$typ] FROM 0 TO rt11ctx$m_typ_end_segment OF
1748 1827 2             SET
1749 1828 2                 [rt11ctx$m_typ_tentative] : %ASCID 'tent';
1750 1829 2                 [rt11ctx$m_typ_empty] : %ASCID 'empty';
1751 1830 2                 [rt11ctx$m_typ_permanent] : %ASCID 'perm';
1752 1831 2                 [rt11ctx$m_typ_end_segment] : %ASCID 'end';
1753 1832 2                 [INRANGE, OUTRANGE] : %ASCID 'unknown';
1754 1833 2             TES);
1755 1834 2
```



```

: 1756      U 1835 2
: 1757      U 1836 2
: 1758      U 1837 2
: 1759      U 1838 2
: 1760      U 1839 2
: 1761      U 1840 2
: 1762      U 1841 2
: 1763      U 1842 2
: 1764      U 1843 2
: 1765      U 1844 2
: 1766      U 1845 2
: 1767      U 1846 2
: 1768      U 1847 2
: 1769      U 1848 2
: 1770      U 1849 1

! Show what we are returning
$debug_print_fao ('!7AS !10<!AF.!AF!> !6UL !AF !XW !XW !XW !XW !XW !XW !XW',
    .ent_typ,
    .ctx [rt11ctx$l_exp_name_len], ctx [rt11ctx$st_exp_name],
    .ctx [rt11ctx$l_exp_type_len], ctx [rt11ctx$st_exp_type],
    .ctx [rt11ctx$w_blocks],
    rt11ctx$st_exp_date, ctx [rt11ctx$st_exp_date],
    .a[0], .a[1], .a[2], .a[3], .a[4], .a[5], .a[6]);

END;

%FI

RETURN true;

END;
```

```

.PSECT EXCH$RT11_PLIT,NOWRT,2

2D 6E 61 4A 00004 MONTHS: .ASCII \Jan-\
2D 62 65 46 00008 .ASCII \Feb-\
2D 72 61 4D 0000C .ASCII \Mar-\
2D 72 70 41 00010 .ASCII \Apr-\
2D 79 61 4D 00014 .ASCII \May-\
2D 6E 75 4A 00018 .ASCII \Jun-\
2D 6C 75 4A 0001C .ASCII \Jul-\
2D 67 75 41 00020 .ASCII \Aug-\
2D 70 65 53 00024 .ASCII \Sep-\
2D 74 63 4F 00028 .ASCII \Oct-\
2D 76 6F 4E 0002C .ASCII \Nov-\
2D 63 65 44 00030 .ASCII \Dec-\
2D 2A 2A 2A 00034 .ASCII \***-\
2D 70 00038 P.AAB: .ASCII \p-\
2D 20 0003A P.AAC: .ASCII \
2E 0003C P.AAD: .ASCII \.
4C 3E 20 65 74 61 64 6F 6E 20 3C 20 0003D P.AAE: .ASCII \ < nodate > \
55 34 21 46 41 21 2D 4C 55 32 21 00048 P.AAG: .ASCII \!2UL-!AF!4UL\
010E000C 00054 P.AAF: .LONG 17694732
00000000 00058 .ADDRESS P.AAG

.EXTRN SYS$FAO

.PSECT EXCH$RT11_CODE,NOWRT,2

07FC 00000

.ENTRY EXCH$RT11_EXPAND_FILENAME, Save R2,R3,R4,- 1721
R5,R6,R7,R8,R9,R10
MOVAB EXCH$UTIL_RADIX50_TO_ASCII, R10
MOVAB P.AAB, R9
SUBL2 #8, SP
MOVL CTX, R6 1771
MOVL #8519924, R2
MOVZWL #452, R1
MOVL R6, R0
JSB EXCH$UTIL_BLOCK_CHECK
PUSHAB 94(R6) 1776
PUSHAB 58(R6)
```

				06	DD	00030	PUSHL	#6		
		6A		03	FB	00032	CALLS	#3, EXCH\$UTIL RADIX50_TO_ASCII		
5E	A6	06		20	3A	00035	LOCC	#32, #6, 94(R6)	1777	
				02	12	0003A	BNEQ	1\$		
				51	D4	0003C	CLRL	R1		
		52		51	D0	0003E	MOVL	R1, CH		
				05	12	00041	BNEQ	2\$	1778	
		51		06	D0	00043	MOVL	#6, R1		
				08	11	00046	BRB	3\$		
	51	50	5E	A6	9E	00048	MOVAB	94(R6), R0		
		52		50	C3	0004C	SUBL3	R0, CH, R1		
		A6	4A	51	D0	00050	MOVL	R1, 74(R6)		
				64	A6	9F	PUSHAB	100(R6)	1780	
			3E	A6	9F	00057	PUSHAB	62(R6)		
				03	DD	0005A	PUSHL	#3		
		6A		03	FB	0005C	CALLS	#3, EXCH\$UTIL RADIX50_TO_ASCII		
64	A6	03		20	3A	0005F	LOCC	#32, #3, 100(R6)	1781	
				02	12	00064	BNEQ	4\$		
		52		51	D4	00066	CLRL	R1		
				51	D0	00068	MOVL	R1, CH		
		51		05	12	0006B	BNEQ	5\$	1782	
				03	D0	0006D	MOVL	#3, R1		
		50	64	08	11	00070	BRB	6\$		
	51	52		A6	9E	00072	MOVAB	100(R6), R0		
		A6	4E	50	C3	00076	SUBL3	R0, CH, R1		
				51	D0	0007A	MOVL	R1, 78(R6)		
			39	A6	95	0007E	TSTB	57(R6)	1786	
		52	A6	06	18	00081	BGEQ	7\$		
				69	B0	00083	MOVW	P.AAB, 82(R6)	1788	
		52	A6	05	11	00087	BRB	8\$		
	50	A6	02	A9	B0	00089	MOVW	P.AAC, 82(R6)	1790	
		4A	4E	A6	C1	0008E	ADDL3	78(R6), 74(R6), R0	1794	
		46	01	A0	9E	00094	MOVAB	1(R0), 70(R6)		
		58		0A	D0	00099	MOVL	#10, R8	1797	
		57	54	A6	9E	0009C	MOVAB	84(R6), R7	1798	
58	20	5E	4A	A6	2C	000A0	MOVC5	74(R6), 94(R6), #32, R8, (R7)		
				67		000A7				
		57	4A	1D	18	000A8	BGEQ	9\$		
		58	4A	A6	C0	000AA	ADDL2	74(R6), R7		
58	20	04	A9	A6	C2	000AE	SUBL2	74(R6), R8		
				01	2C	000B2	MOVC5	#1, P.AAD, #32, R8, (R7)		
				67		000B8				
				0C	18	000B9	BGEQ	9\$		
				57	D6	000BB	INCL	R7		
				58	D7	000BD	DECL	R8		
58	20	64	4E	A6	2C	000BF	MOVC5	78(R6), 100(R6), #32, R8, (R7)		
				67		000C6				
			44	A6	B5	000C7	TSTW	68(R6)	1802	
	67	A6	05	08	12	000CA	BNEQ	10\$		
				08	28	000CC	MOVC3	#11, P.AAE, 103(R6)	1804	
52	44	A6		4F	11	000D2	BRB	12\$		
		05	07B4	00	EF	000D4	EXTZV	#0, #5, 68(R6), YEAR	1807	
		52		C2	9E	000DA	MOVAB	1972(R2), YEAR		
51	44	A6		05	EF	000DF	EXTZV	#5, #5, 68(R6), DAY	1808	
50	45	A6		02	EF	000E5	EXTZV	#2, #5, 69(R6), MON	1809	
				50	D7	000EB	DECL	MON		
		0C		50	D1	000ED	CMPL	MON, #12	1810	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_expand_filename (ctx)

M 1
16-Sep-1984 01:14:37 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:29:07 [EXCHNG.SRC]EXCRT11.B32;1

Page 61
(18)

		03	1B	000F0	BLEQU	11\$	
	50	0C	D0	000F2	MOVL	#12, MON	
	6E	0B	D0	000F3	MOVL	#11, DATE_DESC	1811
04	AE	67	A6	9E 000F8	MOVAB	103(R6), DATE_DESC+4	1812
			52	DD 000FD	PUSHL	YEAR	1813
		CC	A940	DF 000FF	PUSHAL	MONTHS[MON]	
			04	DD 00103	PUSHL	#4	
			51	DD 00105	PUSHL	DAY	
		10	AE	9F 00107	PUSHAB	DATE_DESC	
			7E	D4 0010A	CLRL	-(SP)	
		1C	A9	9F 0010C	PUSHAB	P.AAF	
00000000G	00		07	FB 0010F	CALLS	#7, SYSSFA0	
	0A		50	E8 00116	BLBS	STATUS, 12\$	
00000000G	00		50	DD 00119	PUSHL	STATUS	1815
			01	FB 0011B	CALLS	#1, LIB\$STOP	
				04 00122	RET		
	50		01	D0 00123	MOVL	#1, R0	1848
			04	00126	RET		1849

; Routine Size: 295 bytes, Routine Base: EXCH\$RT11_CODE + 0D0C

```
1772 1850 1 GLOBAL ROUTINE exch$rt11_format_current_date (ent : $ref_bblock) : NOVALUE jsb_r1 = %SBTTL 'exch$rt11_fo
1773 1851 2 BEGIN
1774 1852 3 ++
1775 1853 4
1776 1854 5 FUNCTIONAL DESCRIPTION:
1777 1855 6
1778 1856 7 Format the current date, placing it into the date field of an RT-11 directory entry
1779 1857 8
1780 1858 9 INPUT:
1781 1859 10
1782 1860 11 ent - pointer to the directory entry
1783 1861 12
1784 1862 13 IMPLICIT INPUTS:
1785 1863 14
1786 1864 15 none
1787 1865 16
1788 1866 17 OUTPUTS:
1789 1867 18
1790 1868 19 none
1791 1869 20
1792 1870 21 IMPLICIT OUTPUTS:
1793 1871 22
1794 1872 23 none
1795 1873 24
1796 1874 25 ROUTINE VALUE:
1797 1875 26
1798 1876 27 none
1799 1877 28
1800 1878 29 SIDE EFFECTS:
1801 1879 30
1802 1880 31 none
1803 1881 32 --
1804 1882 33
1805 1883 34 $dbgtrc_prefix ('rt11_format_current_date> ');
1806 1884 35
1807 1885 36 LOCAL
1808 1886 37 timbuf : VECTOR [7, WORD]
1809 1887 38 ;
1810 1888 39
1811 1889 40 BIND
1812 1890 41 year = timbuf [0] : WORD, month = timbuf [1] : WORD, day = timbuf [2] : WORD;
1813 1891 42
1814 1892 43 $numtim (timbuf=timbuf);
1815 1893 44
1816 1894 45 ent [rt11ent$year] = .year - 1972;
1817 1895 46 ent [rt11ent$month] = .month;
1818 1896 47 ent [rt11ent$day] = .day;
1819 1897 48
1820 1898 49 RETURN;
1821 1899 50 END;
```

.EXTRN SYSSNUMTIM

5E

10 C2 00000 EXCH\$RT11 FORMAT_CURRENT_DATE::
SOBL2 #16, SP

: 1850

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_format_current_date (ent)

B 2
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 63
(19)

				51	DD	00003
				7E	D4	00005
			08	AE	9F	00007
	00000000G	00		02	FB	0000A
		50	04	AE	3C	00011
		50	F84C	CO	9E	00015
	51	6E		OC	C1	0001A
61	05	00		50	FO	0001E
	50	6E		OD	C1	00023
60	05	02	06	AE	FO	00027
	50	8E		OC	C1	0002D
60	05	05	04	AE	FO	00031
		5E		10	CO	00037
				05	0003A	

PUSHL	R1
CLRL	-(SP)
PUSHAB	TIMBUF
CALLS	#2, SYSSNUMTIM
MOVZWL	YEAR, R0
MOVAB	-1972(R0), R0
ADDL3	#12, ENT, R1
INSV	R0, #0, #5, (R1)
ADDL3	#13, ENT, R0
INSV	MONTH, #2, #5, (R0)
ADDL3	#12, ENT, R0
INSV	DAY, #5, #5, (R0)
ADDL2	#16, SP
RSB	

1892
1894
1895
1896
1899

: Routine Size: 59 bytes, Routine Base: EXCH\$RT11_CODE + 0E33

```
1823 1900 1 GLOBAL ROUTINE exch$rt11_mount (volb : $ref_bblock) = %SBTTL 'exch$rt11_mount (volb)'
1824 1901 2 BEGIN
1825 1902 3 ++
1826 1903 4
1827 1904 5 FUNCTIONAL DESCRIPTION:
1828 1905 6
1829 1906 7     Perform RT-11 volume specific mount processing
1830 1907 8
1831 1908 9 INPUTS:
1832 1909 10
1833 1910 11     volb - pointer to volb which has been connected to the RT-11 device
1834 1911 12
1835 1912 13 IMPLICIT INPUTS:
1836 1913 14
1837 1914 15     none
1838 1915 16
1839 1916 17 OUTPUTS:
1840 1917 18
1841 1918 19     none
1842 1919 20
1843 1920 21 IMPLICIT OUTPUTS:
1844 1921 22
1845 1922 23     none
1846 1923 24
1847 1924 25 ROUTINE VALUE:
1848 1925 26
1849 1926 27     none
1850 1927 28
1851 1928 29 SIDE EFFECTS:
1852 1929 30
1853 1930 31     none
1854 1931 32 ---
1855 1932 33
1856 1933 34 $dbgtrc_prefix ('rt11_mount> ');
1857 1934 35
1858 1935 36 LOCAL
1859 1936 37     rtv : $ref_bblock,           ! a pointer to the rt11 volb extension
1860 1937 38     seg : $ref_bblock,           ! a pointer to the current directory segment
1861 1938 39     blocks,
1862 1939 40     status
1863 1940 41 ;
1864 1941 42
1865 1942 43 $debug_print_lit ('entry');
1866 1943 44
1867 1944 45 $block_check (1, .volb, volb, 462);
1868 1945 46
1869 1946 47 ! Allocate and initialize our volb extension
1870 1947 48
1871 1948 49 $logic_check (2, (.volb [volb$a_vfmt_specific] EQL 0), 127);
1872 1949 50 rtv = exch$util_vm_allocate (exchblk$a_rt11);
1873 1950 51 CH$FILL (0, rt11$a_end_zero - rt11$a_start_zero, .rtv + rt11$a_start_zero); ! Set part of block to nulls
1874 1951 52 volb [volb$a_vfmt_specific] = .rtv;
1875 1952 53 $block_init (.rtv, rt11);
1876 1953 54
1877 1954 55 ! Read the first part of the volume, the home block on pbn 1
1878 1955 56
1879 1956 57 IF NOT (status = exch$io_rt11_read (.volb, 1, 1, rtv [rt11$a_block_1]))
```

```
1880 1957 2 THEN
1881 1958     RETURN .status;
1882 1959
1883 1960     ! Read the the first directory segment, found on blocks 6 and 7.
1884 1961
1885 1962     IF NOT (status = exch$io_rt11_read (.volb, rt11$k_root_block, 2,
1886 1963         rtv [rt11$t_block_0] + (512 * rt11$k_root_block)))
1887 1964
1888 1965     THEN
1889 1966         RETURN .status;
1890 1967
1891 1968     ! Use the segment get routine to verify this first segment. We temporarily set the present flag because the
1892 1969     get routine expects to see it.
1893 1970
1894 1971     rtv [rt11$v_dir_present] = true;
1895 1972     seg = exch$rt11_dirseg_get (.volb, 1);
1896 1973     ! Get a pointer to the root segment
1897 1974     rtv [rt11$v_dir_present] = false;
1898 1975     ! DIRSEG_GET will have signalled any problems
1899 1976     IF .seg EQL 0
1900 1977     THEN
1901 1978         RETURN exch$_rt11_baddirect;
1902 1979
1903 1980     ! Read in the rest of the directory if it is a multi-segment directory
1904 1981
1905 1982     IF .seg [rt11hdr$w_high_seg] GTRU 1
1906 1983     THEN
1907 1984         BEGIN
1908 1985             LOCAL
1909 1986             blk_cnt, addr;
1910 1987             blk_cnt = 2 * (.seg [rt11hdr$w_high_seg] - 1);
1911 1988             ! Segs are 2 blocks, but one is already in memory
1912 1989             addr = rt11$k_dirseglen + .seg;
1913 1990             ! Get a pointer to space after the root segment
1914 1991             IF NOT (status = exch$io_rt11_read (.volb, rt11$k_root_block+2, .blk_cnt, .addr))
1915 1992             THEN
1916 1993                 RETURN .status;
1917 1994             END;
1918 1995
1919 1996     ! Now we are ok, set the flag that it is present
1920 1997
1921 1998     rtv [rt11$v_dir_present] = true;
1922 1999     ! This means present through HIGH_SEG, not NUM_SEGS
1923 2000
1924 2001     ! Verify the directory
1925 2002
1926 2003     status = exch$rtacp_verify_directory (.volb);
1927 2004
1928 2005     ! Set the volume type string
1929 2006
1930 2007     CH$MOVE (5, UPLIT BYTE ('RT-11'), volb [volb$t_vol_type]);
1931 2008     volb [volb$l_vol_type_len] = 5;
1932 2009
1933 2010     ! Initialize the directory cache to the state of the global /CACHE qualifier
1934 2011
1935 2012     volb [volb$l_dircache] = .exch$a_gbl [excg$v_q_cache];
1936 2013
1937 2014     RETURN .status;
1938 2015
1939 2016     END;
```

31 31 2D 54 52 0005C P.AAH: .ASCII \RT-11\

```
.PSECT EXCH$RT11_CODE,NOWRT,2

.ENTRY EXCH$RT11_MOUNT, Save R2,R3,R4,R5,R6,R7,R8 1900
MOVAB EXCH$IO_RT11_READ, R8
MOVL VOLB, R6 1944
MOVL #68878579, R2
MOVZWL #462, R1
MOVL R6, R0
JSB EXCH$UTIL_BLOCK_CHECK
TSTL 84(R6) 1948
BEQL 1$
MOVZBL #127, -(SP)
PUSHL #1
PUSHL #EXCH$BADLOGIC
CALLS #3, LIB$STOP
MOVZWL #34830, -(SP) 1949
CALLS #1, EXCH$UTIL_VM_ALLOCATE
MOVL R0, RTV
MOVAB 12(RTV), R3 1950
CLRW (R3)
MOVL RTV, 84(R6) 1951
MOVW #-30706, 8(RTV) 1952
MNEGB #11, 10(RTV)
PUSHAB 526(RTV) 1956
PUSHL #1
PUSHL #1
PUSHL R6
CALLS #4, EXCH$IO_RT11_READ
MOVL R0, STATUS
BLBC STATUS, 4$
PUSHAB 3086(RTV) 1963
PUSHL #2 1962
PUSHL #6
PUSHL R6
CALLS #4, EXCH$IO_RT11_READ
MOVL R0, STATUS
BLBC STATUS, 4$
BISB2 #1, (R3) 1970
PUSHL #1 1971
PUSHL R6
CALLS #2, EXCH$RT11_DIRSEG_GET
BICB2 #1, (R3) 1972
TSTL SEG 1973
BNEQ 2$
MOVL #EXCH$_RT11_BADDIRECT, R0 1975
RET
CMPW 4(SEG), #1 1979
BLEQU 3$
MOVZWL 4(SEG), R1 1984
DECL R1
MULL2 #2, BLK CNT
MOVAB 1024(R0), ADDR 1985
```


EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_mount (volb)

F 2
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 67
(20)

				50	DD	000B2	PUSHL	ADDR		1986
				51	DD	000B4	PUSHL	BLK_CNT		
				08	DD	000B6	PUSHL	#8		
				56	DD	000B8	PUSHL	R6		
		68		04	FB	000BA	CALLS	#4, EXCH\$IO_RT11_READ		
		57		50	DO	000BD	MOVL	R0, STATUS		
		24		57	FF	000C0	BLBC	STATUS, 48		
		63		01	88	000C3	BISB2	#1, (R3)		1993
				56	DD	000C6	PUSHL	R6		1997
		00000000G	EF	01	FB	000C8	CALLS	#1, EXCH\$RTACP_VERIFY_DIRECTORY		
			57	50	DO	000CF	MOVL	R0, STATUS		
	5D	A6	0000'	05	28	000D2	MOVCS	#5, P,AAH, 93(R6)		2001
			59	05	DO	000D9	MOVL	#5, 89(R6)		2002
50	A6	00000000G	FF	01	EF	000DD	EXTZV	#1, #1, @EXCH\$A_GBL, 80(R6)		2006
				50	DO	000E7	MOVL	STATUS, R0		2008
				57	04	000EA	RET			2009

; Routine Size: 235 bytes, Routine Base: EXCH\$RT11_CODE + 0E6E

```
1934 2010 1 GLOBAL ROUTINE exch$rt11_open_file = %SBTTL 'exch$rt11_open_file'
1935 2011 2 BEGIN
1936 2012 3 ++
1937 2013 4
1938 2014 5 FUNCTIONAL DESCRIPTION:
1939 2015 6
1940 2016 7     Perform RT-11 volume specific open processing
1941 2017 8
1942 2018 9 INPUT:
1943 2019 10
1944 2020 11     none
1945 2021 12
1946 2022 13 IMPLICIT INPUTS:
1947 2023 14
1948 2024 15     copy work area
1949 2025 16
1950 2026 17 OUTPUTS:
1951 2027 18
1952 2028 19     none
1953 2029 20
1954 2030 21 IMPLICIT OUTPUTS:
1955 2031 22
1956 2032 23     filb - receive info pertaining to the open file
1957 2033 24
1958 2034 25 ROUTINE VALUE:
1959 2035 26
1960 2036 27     true if able to open a file, false otherwise
1961 2037 28
1962 2038 29 SIDE EFFECTS:
1963 2039 30
1964 2040 31     none
1965 2041 32 --
1966 2042 33
1967 2043 34 $dbgtrc_prefix ('rt11_open_file> ');
1968 2044 35
1969 2045 36 LOCAL
1970 2046 37     out_filb                : $ref_bblock,
1971 2047 38     status
1972 2048 39     ;
1973 2049 40
1974 2050 41 BIND
1975 2051 42     copy      = exch$a_gbl [excg$a_copy_work]      : $ref_bblock,
1976 2052 43     inp_filb  = copy [copy$a_inp_filb]             : $ref_bblock,
1977 2053 44     ctx       = inp_filb [fi[b$a_context]          : $ref_bblock,
1978 2054 45     namb      = inp_filb [filb$a_assoc_namb]       : $ref_bblock,
1979 2055 46     nam_nam   = namb [namb$a_name]                 : $desc_block,
1980 2056 47     nam_typ   = namb [namb$a_type]                 : $desc_block,
1981 2057 48     volb      = inp_filb [fi[b$a_assoc_volb]       : $ref_bblock,
1982 2058 49     inp_namb  = copy [copy$a_inp_namb]             : $ref_bblock
1983 2059 50     ;
1984 2060 51
1985 2061 52 $debug_print_lit ('entry');
1986 2062 53
1987 2063 54 $block_check (2, .inp_filb, filb, 463);
1988 2064 55 $block_check (2, .namb, namb, 464);
1989 2065 56 $block_check (2, .volb, volb, 465);
1990 2066 57
```

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_open_file

H 2
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 BLISS-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 69
(21)

```
: 1991      2067 2  | Make sure that the output filb points to a valid filb
: 1992      2068 2  |
: 1993      2069 2  | IF (out_filb = .copy [copy$a_out_filb]) EQL 0
: 1994      2070 2  | THEN
: 1995      2071 2  |     out_filb = .inp_filb;
: 1996      2072 2  | $block_check (2, .out_filb, filb, 472);
```

```
1998 2073 2 ! If the context pointer is null, then allocate and initialize it.
1999 2074 2
2000 2075 2 IF .ctx EQL 0
2001 2076 2 THEN
2002 2077 2     ctx = exch$util_rt11ctx_allocate (.volb, .inp_filb) ! Create an RT11 context block
2003 2078 2
2004 2079 2 ! If non-null, we are doing a subsequent lookup in a wildcard search
2005 2080 2
2006 2081 2 ELSE
2007 2082 2 BEGIN
2008 2083 2
2009 2084 2     ! If not wildcard, then we must be done
2010 2085 2     !
2011 2086 2     IF NOT (.namb [namb$v_wild_name] OR .namb [namb$v_wild_type])
2012 2087 2     THEN
2013 2088 2         RETURN false;
2014 2089 2
2015 2090 2     $block_check (2, .ctx, rt11ctx, 446);
2016 2091 2
2017 2092 2 END;
2018 2093 2
2019 2094 2 ! Make sure that we haven't crossed signals someplace
2020 2095 2
2021 2096 2 $logic_check (4, (.ctx [rt11ctx$a_assoc_filb] EQL .inp_filb), 128);
2022 2097 2 $logic_check (4, (.ctx [rt11ctx$a_assoc_volb] EQL .volb), 129);
2023 2098 2
2024 2099 2 ! We assume that the file name and type fields in the namb are adjacent. If they aren't, the next call to
2025 2100 2 exch$rtacp_find_file will choke.
2026 2101 2
2027 2102 2 $logic_check (3, (.nam_typ [dsc$a_pointer] EQL (.nam_nam [dsc$w_length] + .nam_nam [dsc$a_pointer])), 154);
```



```
2029 2103 IF (
2030 2104     IF .copy [copy$reopen_input]           ! If we are retrying, then reuse the context block
2031 2105     THEN
2032 2106         1
2033 2107     ELSE
2034 2108         ! Otherwise skip to the next file
2035 2109         exch$rtacp_find_file (.ctx, .nam_nam [dsc$a_pointer], .nam_nam [dsc$w_length] + .nam_typ [dsc$w_leng
2036 2110     )
2037 2111     THEN
2038 2112         BEGIN
2039 2113         ! Do not find a .BAD file unless it is explicitly specified
2040 2114         IF .ctx [rt11ctx$w_filetype] EQL r50_bad
2041 2115         THEN
2042 2116             IF .inp_namb [namb$w_wild_name]
2043 2117                 ! If the found file was not explicitly named
2044 2118                 OR
2045 2119                 ! then skip to the next file by calling
2046 2120                 ! ourselves again
2047 2121                 .inp_namb [namb$w_wild_type]
2048 2122             THEN
2049 2123                 RETURN exch$rt11_open_file ();
2050 2124             ! Create the result name string in the filb
2051 2125             inp_filb [filb$l_result_name_len] = .volb [volb$l_vol_ident_len]      ! Length of volume ident
2052 2126             + .ctx [rt11ctx$l_exp_fullname_len];      ! plus rt fullname
2053 2127             $logic check (3, (.inp_filb [filb$l_result_name_len] LEQU filb$l_result_name), 130);
2054 2128             CH$COPY (.volb [volb$l_vol_ident_len], volb [volb$l_vol_ident],      ! Volume name
2055 2129                 .ctx [rt11ctx$l_exp_fullname_len], ctx [rt11ctx$l_exp_fullname],
2056 2130                 0, filb$l_result_name, inp_filb [filb$l_result_name]);
2057 2131             $debug_print_fao ('Found "AF"', .inp_filb [filb$l_result_name_len], inp_filb [filb$l_result_name]);
2058 2132             ! Define a record stream for this file
2059 2133             !
2060 2134             ctx [rt11ctx$l_cur_byte] = 0;
2061 2135             ctx [rt11ctx$l_cur_block] = .ctx [rt11ctx$l_start_block];      ! Context is the first byte in
2062 2136             ! the first block of the file
2063 2137             ctx [rt11ctx$l_eof_block] = .ctx [rt11ctx$l_start_block] + .ctx [rt11ctx$w_blocks] - 1;
2064 2138             inp_filb [filb$l_block_count] = .ctx [rt11ctx$w_blocks];      ! Put the size in the filb
2065 2139             ! No valid record or length
2066 2140             inp_filb [filb$a_record] = 0;
2067 2141             inp_filb [filb$l_record_len] = 0;
2068 2142             ! Make sure that the record format in the filb is correct
2069 2143             !
2070 2144             exch$cmd_fetch_recfmt_implied (.inp_filb, ctx [rt11ctx$l_exp_type]);
2071 2145             ! For RT-11 we can treat block transfer mode as fixed 512
2072 2146             !
2073 2147             IF .out_filb [filb$b_transfer_mode] EQL filb$b_k_xfrm_block
2074 2148                 OR
2075 2149                 .inp_filb [filb$b_transfer_mode] EQL filb$b_k_xfrm_block
2076 2150             THEN
2077 2151                 BEGIN
2078 2152                     inp_filb [filb$b_rec_format] = filb$b_rfmt_fixed;
2079 2153                     inp_filb [filb$l_fixed_len] = 512;
2080 2154                 END;
2081 2155             ! Clear all the flags except the ones we want by writing the masks into the longword
2082 2156             !
2083 2157             !
2084 2158             !
2085 2159             !
```

```

2086      ctx [rt11ctx$l_flags] = rt11ctx$m_stream_active;      ! A record stream is currently active
2087      inp_filb [filb$v_files_found] = true;                  ! One or more files have been opened
2088
2089      ! Set up the i/o and record buffer (for when we can't use locate mode)
2090
2091      IF .ctx [rt11ctx$a_buffer] EQL 0                        ! If doing wildcards buffer might be there
2092      THEN
2093          ctx [rt11ctx$a_buffer] = exch$util_vm_allocate (ctx$k_buffer_length);
2094
2095      ! Set the block pointers so that we will advance the buffer on the first get
2096
2097      ctx [rt11ctx$l_buf_base_block] = .ctx [rt11ctx$l_start_block];
2098      ctx [rt11ctx$l_buf_high_block] = .ctx [rt11ctx$l_start_block] - 1;
2099
2100      ! Save the addresses of our routines for this volume and record format.
2101
2102      inp_filb [filb$a_close_routine] = exch$rt11_close_file;
2103      inp_filb [filb$a_put_routine] = 0;
2104      inp_filb [filb$a_get_routine] = exch$pdp_get;
2105
2106      RETURN true;                                           ! True means its open
2107      END;
2108
2109      ! If no files were found, then scream and shout
2110
2111      IF NOT .inp_filb [filb$v_files_found]
2112      THEN
2113          BEGIN
2114              REGISTER
2115                  fao_desc = 0 : $ref_bblock;
2116
2117              ! Concatenate the volume name to the file name and type fields
2118
2119              fao_desc = exch$util_fao_buffer (%ASCII '!AF!AF', .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident],
2120                  .nam_nam [dsc$a_length] + .nam_typ [dsc$a_length], .nam_nam [dsc$a_pointer]);
2121              $exch signal (exch$_file_not_found, 1, .fao_desc);
2122              RETURN rms$_fnf;
2123          END;
2124
2125      ! Normal exit, return a 0
2126
2127      RETURN 0;
2128      END;
2129

```

```

00 00 46 41 21 46 41 21 00061
010E0006 00064 P.AAJ:
00000000 0006C P.AAI:
00070

.PSECT EXCH$RT11_PLIT,NOWRT,2
.BKLB 3
.ASCII \!AF!AF\<0><0>
.LONG 17694726
.ADDRESS P.AAJ

.EXTRN EXCH$_FILENOTFOUND
.PSECT EXCH$RT11_CODE,NOWRT,2

```

			OFFC	00000	.ENTRY	EXCH\$RT11_OPEN_FILE, Save R2,R3,R4,R5,R6,-			
						R7,R8,R9,R10,RT1	2010		
					SUBL2	#4, SP			
50	00000000G	SE	04	C2	00002	ADDL3	#4, EXCH\$A_GBL, R0	2051	
		EF	04	C1	00005	MOVL	(R0), R3	2052	
		53	60	DO	0000D	MOVL	60(R3), R7	2053	
		57	3C	A3	DO	00010	MOVL	24(R7), R9	2055
		59	18	A7	DO	00014	PUSHAB	80(R9)	
			50	A9	9F	00018	MOVL	#56295674, R2	2063
		52	035B00FA	8F	DO	0001B	MOVZWL	#463, R1	
		51	01CF	8F	3C	00022	MOVL	R7, R0	
		50		57	DO	00027	JSB	EXCH\$UTIL_BLOCK_CHECK	
			00000000G	EF	16	0002A	MOVL	#17432823, R2	2064
		52	010A00F7	8F	DO	00030	MOVZWL	#464, R1	
		51	01D0	8F	3C	00037	MOVL	R9, R0	
		50		59	DO	0003C	JSB	EXCH\$UTIL_BLOCK_CHECK	
			00000000G	EF	16	0003F	MOVL	28(R7), R8	2065
		58	1C	A7	DO	00045	MOVL	#68878579, R2	
		52	041B00F3	8F	DO	00049	MOVZWL	#465, R1	
		51	01D1	8F	3C	00050	MOVL	R8, R0	
		50		58	DO	00055	JSB	EXCH\$UTIL_BLOCK_CHECK	
			00000000G	EF	16	00058	MOVL	68(R3), OUT_FILB	2069
		58	44	A3	DO	0005E	BNEQ	1\$	
				03	12	00062	MOVL	R7, OUT_FILB	2071
		58		57	DO	00064	MOVL	#56295674, R2	2072
		52	035B00FA	8F	DO	00067	MOVZWL	#472, R1	
		51	01D8	8F	3C	0006E	MOVL	OUT_FILB, R0	
		50		58	DO	00073	JSB	EXCH\$UTIL_BLOCK_CHECK	
			00000000G	EF	16	00076	TSTL	32(R7)	2075
			20	A7	D5	0007C	BNEQ	2\$	
				11	12	0007F	PUSHL	R7	2077
				57	DD	00081	PUSHL	R8	
				58	DD	00083	CALLS	#2, EXCH\$UTIL_RT11CTX_ALLOCATE	
		00000000G	EF	02	FB	00085	MOVL	R0, 32(R7)	
		20	A7	50	DO	0008C	BRB	4\$	
				23	11	00090	BBS	#1, 108(R9), 3\$	2086
08	6C	A9		01	E0	00092	BBS	#2, 108(R9), 3\$	
03	6C	A9		02	E0	00097	BRW	13\$	
			0147	31	0009C		MOVL	#8519924, R2	2090
		52	008200F4	8F	DO	0009F	MOVZWL	#446, R1	
		51	01BE	8F	3C	000A6	MOVL	32(R7), R0	
		50	20	A7	DO	000AB	JSB	EXCH\$UTIL_BLOCK_CHECK	
			00000000G	EF	16	000AF	BBS	#2, 52(R3), 5\$	2104
22	34	A3		02	E0	000B5	MOVZWL	80(SP), R0	2108
		50	00	BE	3C	000BA	MOVZWL	88(R9), R1	
		51	58	A9	3C	000BE	PUSHAB	(R1)[R0]	
			6140	9F	000C2		ADDL3	#4, 4(SP), R2	
52	04	AE		04	C1	000C5	PUSHL	(R2)	
			20	62	DD	000CA	PUSHL	32(R7)	
				A7	DD	000CC	CALLS	#3, EXCH\$RTACP_FIND_FILE	
		00000000G	EF	03	FB	000CF	BLBS	R0, 5\$	
		03		50	E8	000D6	BRW	12\$	
				00CA	31	000D9	MOVL	32(R7), R6	2115
		56	20	A7	DO	000DC	CMPL	62(R6), #3244	
	OCAC	8F	3E	A6	B1	000E0	BNEQ	7\$	
				14	12	000E6	MOVL	64(R3), R0	2117
		50	40	A3	DO	000E8			

04	AE	00	69	A8	65	A8	2C	0010D	MOVCS	101(R8), 105(R8), #0, 4(SP), (R10)	2130
04	AE	00	54	A6	46	A6	2C	00121	MOVCS	70(R6), 84(R6), #0, 4(SP), (R10)	2136
			1C	A6	72	A6	DD	0012D	MOVL	114(R6), 28(R6)	2137
				50	40	A6	3C	00132	MOVZWL	54(R6), R0	2138
				50	72	A6	CO	00136	ADDL2	114(R6), R0	
			20	A6	FF	A0	9E	0013A	MOVAB	-1(R0), 32(R6)	2139
			3E	A7	40	A6	3C	0013F	MOVZWL	64(R6), 62(R7)	2141
					42	A7	7C	00144	CLRQ	66(R7)	2145
					64	A6	9F	00147	PUSHAB	100(R6)	
						57	DD	0014A	PUSHL	R7	
		00000000G		EF		02	FB	0014C	CALLS	#2, EXCH\$CMD_FETCH_RECMT_IMPLIED	2149
				01	29	AB	91	00153	CMPB	41(OUT_FILB), #1	
						06	13	00157	BEQL	9\$	2151
				01	29	A7	91	00159	CMPB	41(R7), #1	
						0A	12	0015D	BNEQ	10\$	2154
			28	A7	02	90	0015F	9\$:	MOVAB	#2, 40(R7)	2155
			35	A7	8F	3C	00163	10\$:	MOVZWL	#512, 53(R7)	2160
			28	A6	01	DD	00169		MOVL	#1, 40(R6)	2161
			2B	A7	08	88	0016D		BISB2	#8, 43(R7)	2165
					18	A6	D5	00171	TSTL	24(R6)	
						10	12	00174	BNEQ	11\$	2167
				7E	1800	8F	3C	00176	MOVZWL	#6144, -(SP)	
		00000000G		EF		01	FB	0017B	CALLS	#1, EXCH\$UTIL_VM_ALLOCATE	2171
			18	A6	50	DD	00182	11\$:	MOVL	R0, 24(R6)	2172
			2C	A6	72	A6	DD	00186	MOVL	114(R6), 44(R6)	2176
30	A6		72	A6	01	C3	0018B		SUBL3	#1, 114(R6), 48(R6)	2177
			4A	A7	CF	9E	00191		MOVAB	EXCH\$RT11_CLOSE_FILE, 74(R7)	2178
					56	A7	D4	00197	CLRQ	86(R7)	2180
			52	A7	00000000G	EF	9E	0019A	MOVAB	EXCH\$PDP_GET, 82(R7)	
				50		01	DD	001A2	MOVL	#1, R0	2185
						04	001A5	12\$:	RET		2194
			3B	A7	03	E0	001A6		BBS	#3, 43(R7), 13\$	
			52	6E	04	C1	001AB		ADDL3	#4, (SP), R2	
					62	DD	001AF		PUSHL	(R2)	
				50	BE	3C	001B1		MOVZWL	24(SP), R0	
				51	A9	3C	001B5		MOVZWL	88(R9), R1	
					6140	9F	001B9		PUSHAB	(R1)(R0)	2193
					69	A8	9F	001BC	PUSHAB	105(R8)	
					65	A8	DD	001BF	PUSHL	101(R8)	
					0000'	CF	9F	001C2	PUSHAB	P, AAI	
		00000000G		EF		05	FB	001C6	CALLS	#5, EXCH\$UTIL_FAO_BUFFER	2195
						50	DD	001CD	PUSHL	FAD_DESC	
						01	DD	001CF	PUSHL	#1	
					00000000G	8F	DD	001D1	PUSHL	#EXCH\$_FILENOTFOUND	

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_open_file

N 2
16-Sep-1984 01:14:37 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:29:07 [EXCHNG.SRC]EXCRT11.B32;1

Page 75
(23)

00000000G	00	03	FB	001D7	CALLS	#3	LIB\$SIGNAL
	50	8F	D0	001DE	MOVL	#98962,	R0
			04	001E5	RET		
		50	D4	001E6	CLRL	R0	
			04	001E8	RET		

13\$:

: 2196
: 2203
:

; Routine Size: 489 bytes, Routine Base: EXCH\$RT11_CODE + 0F59

```
2131 2204 1 GLOBAL ROUTINE exch$rt11_write_cleanup (volb : $ref_bblock) : NOVALUE = %SBTTL 'exch$rt11_write_cleanup (vol
2132 2205 2 BEGIN
2133 2206 2 ++
2134 2207 2
2135 2208 2 FUNCTIONAL DESCRIPTION:
2136 2209 2
2137 2210 2 Finish writing to the volume. Clear file marks and flush caches.
2138 2211 2
2139 2212 2 INPUTS:
2140 2213 2
2141 2214 2 volb - pointer to volb which has been connected to the RT-11 device
2142 2215 2
2143 2216 2 IMPLICIT INPUTS:
2144 2217 2
2145 2218 2 none
2146 2219 2
2147 2220 2 OUTPUTS:
2148 2221 2
2149 2222 2 none
2150 2223 2
2151 2224 2 IMPLICIT OUTPUTS:
2152 2225 2
2153 2226 2 none
2154 2227 2
2155 2228 2 ROUTINE VALUE:
2156 2229 2
2157 2230 2 none
2158 2231 2
2159 2232 2 SIDE EFFECTS:
2160 2233 2
2161 2234 2 error conditions will be signaled
2162 2235 2 --
2163 2236 2
2164 2237 2 $subgrc_prefix ('rt11_write_cleanup> ');
2165 2238 2
2166 2239 2 $trace_print_fao ('entry - volb !XL', .volb);
2167 2240 2
2168 2241 2 exch$rt11_zero_marks (.volb); ! Clear all the file marks
2169 2242 2
2170 2243 2 exch$rt11_dircache_stop (.volb); ! Clear caching and flush the directory
2171 2244 2
2172 2245 2 RETURN;
2173 2246 2 END;
```

				0000	00000
		04	AC	DD	00002
0000V	CF		01	FB	00005
		04	AC	DD	0000A
F90F	CF		01	FB	0000D
				04	00012

.ENTRY	EXCH\$RT11_WRITE_CLEANUP, Save nothing	2204
PUSHL	VOLB	2241
CALLS	#1, EXCH\$RT11_ZERO_MARKS	
PUSHL	VOLB	2243
CALLS	#1, EXCH\$RT11_DIRCACHE_STOP	
RET		2246

; Routine Size: 19 bytes, Routine Base: EXCH\$RT11_CODE + 1142

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_write_cleanup (volb)

^C₃
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 77
(24)

EX
VO

```
2175 2247 1 GLOBAL ROUTINE exch$rt11_write_prepare (volb : $ref_bblock) : NOVALUE = %SBTTL 'exch$rt11_write_prepare (vol
2176 2248 2 BEGIN
2177 2249 3 ++
2178 2250 4
2179 2251 5 FUNCTIONAL DESCRIPTION:
2180 2252 6
2181 2253 7     Prepare to write to the volume. Set up caches and clear file marks.
2182 2254 8
2183 2255 9 INPUTS:
2184 2256 10
2185 2257 11     volb - pointer to volb which has been connected to the RT-11 device
2186 2258 12
2187 2259 13 IMPLICIT INPUTS:
2188 2260 14
2189 2261 15     none
2190 2262 16
2191 2263 17 OUTPUTS:
2192 2264 18
2193 2265 19     none
2194 2266 20
2195 2267 21 IMPLICIT OUTPUTS:
2196 2268 22
2197 2269 23     none
2198 2270 24
2199 2271 25 ROUTINE VALUE:
2200 2272 26
2201 2273 27     none
2202 2274 28
2203 2275 29 SIDE EFFECTS:
2204 2276 30
2205 2277 31     error conditions will be signaled
2206 2278 32 --
2207 2279 33
2208 2280 34 $dbgtrc_prefix ('rt11_write_prepare> ');
2209 2281 35
2210 2282 36 $trace_print_fao ('entry - volb !XL', .volb);
2211 2283 37
2212 2284 38 exch$rt11_dircache_start (.volb);           ! Start caching on the directory
2213 2285 39
2214 2286 40 exch$rt11_zero_marks (.volb);               ! Clear all the file marks
2215 2287 41
2216 2288 42 RETURN;
2217 2289 43 END;
```

			04	AC	DD	00002
F86F	CF		01	FB	00005	
			04	AC	DD	0000A
0000V	CF		01	FB	0000D	
				04	00012	

ENTRY	EXCH\$RT11_WRITE_PREPARE, Save nothing	2247
PUSHL	VOLB	2284
CALLS	#1, EXCH\$RT11_DIRCACHE_START	
PUSHL	VOLB	2286
CALLS	#1, EXCH\$RT11_ZERO_MARKS	
RET		2289

; Routine Size: 19 bytes, Routine Base: EXCH\$RT11_CODE + 1155

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_write_prepare (volb)

E 3
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 79
(25)

EX
VO

```
2219 1 GLOBAL ROUTINE exch$rt11_zero_marks (volb : $ref_bblock) : NOVALUE = %SBTTL 'exch$rt11_zero_marks (volb)'  
2220 BEGIN  
2221 ++  
2222  
2223 FUNCTIONAL DESCRIPTION:  
2224  
2225 Clear the file marks in every entry in the directory. EXCHANGE flags the JOB byte (rt11ent$b_job) o  
2226 directory entry while it is entering files. The flags are cleared before and after each COPY comman  
2227 thereby giving us the means to recognize that we might have to delete a file which we just created.  
2228  
2229 INPUTS:  
2230  
2231 volb - pointer to volb which has been connected to the RT-11 device  
2232  
2233 IMPLICIT INPUTS:  
2234  
2235 none  
2236  
2237 OUTPUTS:  
2238  
2239 none  
2240  
2241 IMPLICIT OUTPUTS:  
2242  
2243 none  
2244  
2245 ROUTINE VALUE:  
2246  
2247 none  
2248  
2249 SIDE EFFECTS:  
2250  
2251 error conditions will be signaled  
2252  
2253 --  
2254 $dbgtrc_prefix ('exch$rt11_zero_marks> ');  
2255  
2256 LOCAL  
2257 seg : $ref_bblock, ! a pointer to the current directory segment  
2258 cur : $ref_bblock, ! a pointer to the current directory entry  
2259 seg_num, ! current segment number  
2260 ent_len ! length of a directory entry  
2261 ;  
2262  
2263 $trace_print_fao ('entry - volb !XL', .volb);  
2264  
2265 $block_check (2, .volb, volb, 459);  
2266 $logic_check (2, (.volb [volb$v_write]), 199); ! We shouldn't get this far if we aren't supposed to write t
```

```
2268 2338 2 ! Loop through all the directory entries to clear the mark flag.
2269 2339 2
2270 2340 2 seg_num = 1; ! Start with the first directory segment
2271 2341 2 WHILE .seg_num NEQ 0
2272 2342 2 DO
2273 2343 2 BEGIN
2274 2344 2 ! Get a pointer to the current segment, return if error
2275 2345 2
2276 2346 2 seg = exch$rt11_dirseg_get (.volb, .seg_num);
2277 2347 2 $logic_check (2, (.seg NEQ 0), 197);
2278 2348 2 $ent_len = rt11ent$k_length + .seg [rt11hdr$w_extra_bytes]; ! Actually the same for all segments
2279 2349 2
2280 2350 2 ! Get a pointer to the first directory entry
2281 2351 2
2282 2352 2 cur = .seg + rt11hdr$k_length;
2283 2353 2
2284 2354 2 ! Look through the segment
2285 2355 2
2286 2356 2 WHILE (.cur LSSU (.seg + rt11$k_dirseglen))
2287 2357 2 DO
2288 2358 2 BEGIN
2289 2359 2
2290 2360 2 CASE .cur [rt11ent$v_type] FROM 0 TO rt11ent$m_typ_end_segment OF
2291 2361 2 SET
2292 2362 2 [rt11ent$m_typ_tentative, rt11ent$m_typ_permanent, rt11ent$m_typ_empty] :
2293 2363 2 BEGIN
2294 2364 2
2295 2365 2 ! If the marker isn't clear, clear it and remember that the segment has been changed
2296 2366 2
2297 2367 2 IF .cur [rt11ent$b_job] NEQ 0
2298 2368 2 THEN
2299 2369 2 BEGIN
2300 2370 2 cur [rt11ent$b_job] = 0;
2301 2371 2 exch$rt11_dirseg_put (.volb, .seg_num); ! Caching is on, this won't give us an I/O n
2302 2372 2 END;
2303 2373 2
2304 2374 2 END;
2305 2375 2
2306 2376 2 [rt11ent$m_typ_end_segment] :
2307 2377 2 EXITLOOP;
2308 2378 2
2309 2379 2 [INRANGE, OUTRANGE] :
2310 2380 2
2311 2381 2 TES;
2312 2382 2
2313 2383 2 cur = .cur + .ent_len; ! Skip to the next entry
2314 2384 2
2315 2385 2 END;
2316 2386 2
2317 2387 2 seg_num = .seg [rt11hdr$w_next_seg]; ! Skip to the next segment
2318 2388 2
2319 2389 2 END;
2320 2390 2
2321 2391 2 RETURN;
2322 2392 2
2323 2393 2 END;
```

				07FC 00000	.ENTRY	EXCH\$RT11 ZERO_MARKS, Save R2,R3,R4,R5,R6,-	
						R7,R8,R9,R10	2290
					MOVAB	LIB\$STOP, R10	
					MOVL	#EXCH\$BADLOGIC, R9	
					MOVL	VOLB, R5	2336
					MOVL	#68878579, R2	
					MOVZWL	#459, R1	
					MOVL	R5, R0	
					JSB	EXCH\$UTIL_BLOCK_CHECK	
					BBS	#5, 72(R5), 1\$	2337
					MOVZBL	#199, -(SP)	
					PUSHL	#1	
					PUSHL	R9	
					CALLS	#3, LIB\$STOP	
					MOVL	#1, SEG_NUM	2340
					BEQL	9\$	2341
					PUSHL	SEG_NUM	2347
					PUSHL	R5	
					CALLS	#2, EXCH\$RT11_DIRSEG_GET	
					MOVL	R0, SEG	
					BNEQ	3\$	2348
					MOVZBL	#197, -(SP)	
					PUSHL	#1	
					PUSHL	R9	
					CALLS	#3, LIB\$STOP	
					MOVZWL	6(SEG), ENT_LEN	2349
					ADDL2	#14, ENT_LEN	
					MOVAB	10(R3), CUR	2353
					MOVAB	1024(R3), R6	2357
					CMPL	CUR, R6	
					BGEQU	8\$	
					EXTZV	#0, #4, 1(CUR), R8	2361
					CASEL	R8, #0, #8	
					.WORD	7\$-5\$, -	
						6\$-5\$, -	
						6\$-5\$, -	
						7\$-5\$, -	
						6\$-5\$, -	
						7\$-5\$, -	
						7\$-5\$, -	
						7\$-5\$, -	
						8\$-5\$	
					BRB	7\$	
					TSTB	11(CUR)	2368
					BEQL	7\$	
					CLRB	11(CUR)	2371
					PUSHL	SEG_NUM	2372
					PUSHL	R5	
					CALLS	#2, EXCH\$RT11_DIRSEG_PUT	
					ADDL2	ENT_LEN, CUR	2384
					BRB	4\$	2357
					MOVZWL	2(SEG), SEG_NUM	2388
					BRB	2\$	2341

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_zero_marks (volb)

1 3
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 83
(27)

04 000A6 9\$: RET

; 2393

; Routine Size: 167 bytes, Routine Base: EXCH\$RT11_CODE + 1168

EX
VO

EXCH\$RT11
V04-000

RT11 file and directory routines
exch\$rt11_zero_marks (volb)

J 3
16-Sep-1984 01:14:37
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC]EXCRT11.B32;1

Page 84
(28)

: 2325
: 2326
2394 1 END
2395 0 ELUDOM

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
EXCH\$RT11_CODE	4623 NOVEC,NOWRT, RD	EXE,NOSHR, LCL, REL, CON,NOPI,ALIGN(2)
EXCH\$RT11_PLIT	116 NOVEC,NOWRT, RD	EXE,NOSHR, LCL, REL, CON,NOPI,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	22	0	1000	00:01.9
\$255\$DUA28:[EXCHNG.OBJ]EXCLIB.L32;1	1151	188	16	79	00:01.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:EXCRT11/OBJ=OBJ\$:EXCRT11 MSRC\$:EXCRT11/UPDATE=(ENH\$:EXCRT11)

: Size: 4623 code + 116 data bytes
: Run Time: 01:22.6
: Elapsed Time: 04:13.3
: Lines/CPU Min: 1739
: Lexemes/CPU-Min: 23430
: Memory Used: 374 pages
: Compilation Complete

0162 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0163 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

